INSTITUTO LATINO-
AMERICANO DE TECNOLOGIA,
INFRAESTRUTURA E TERRITÓRIO (ILATIT)

PROGRAMA LIVRE PARA DIMENSIONAMENTO DE CONSOLO PRÉ-MOLDADO DE CONCRETO ARMADO

ABEL ISAAC ARÉVALOS SEQUEIRA

Foz do Iguaçu
2025

INSTITUTO LATINO-AMERICANO DE TECNOLOGIA, INFRAESTRUTURA E TERRITÓRIO (ILATIT)

PROGRAMA LIVRE PARA DIMENSIONAMENTO DE CONSOLO PRÉ-MOLDADO DE CONCRETO ARMADO

ABEL ISAAC ARÉVALOS SEQUEIRA

Trabalho de Conclusão de Curso apresentado à Banca Examinadora do Curso de Engenharia Civil de Infraestrutura da UNILA, como parte dos requisitos para a obtenção do Grau de Bacharel em Engenharia Civil de Infraestrutura

Orientador: Profº. Dr. Aref Kalilo Lima Kzam

Foz do Iguaçu
2025

AGRADECIMENTOS

A Deus, pelo acompanhamento na solidão e quietude das noites de estudo, pela força para resistir ainda nos momentos onde todo se quebra e ninguém pode ajudar. Pela escuta contínua mesmo sem merecer.

A Minha Mãe pelo apoio incondicional ainda quando eu não acreditei no processo. Pela confiança cega na minha capacidade de vencer os obstáculos desta jornada. E o mais importante, sua perseverança nas orações.

A meu irmão Ever, por ser aquela pessoa que caminhou do meu lado em cada momento, agindo como válvula de saída para os momentos de cansaço e auxiliar como bombeiro de na resolução de diversas situações.

Ao Professor Aref, pela paciência e compressão. Por nunca desistir e ainda insistir na minha formação. Pelas esporádicas conversas que consegui aproveitar.

RESUMO

Nos últimos anos, no Brasil, viu-se um aumento pela opção do sistema construtivo de elementos pré-moldados de concreto armado. Isto devido maioritariamente à velocidade de construção. Porém, existem algumas dificuldades inerentes ao sistema construtivo. Esses problemas são observados por exemplo nas ligações viga-pilar feitas por consolos e são abordados por diversos autores e consequentemente diversas vias de solução são exploradas. Então, surge a necessidade de uma ferramenta que automatize e agilize o dimensionamento para o avanço dessas pesquisas. Assim, o trabalho a continuação visa fornecer a implementação de um algoritmo computacional para o dimensionamento de consolos pré-moldados de concreto armado, atendendo a Norma Técnica Brasileira 9062 – 2017 e além disso considere diferentes tipos de consolos (segundo a geometria). Isto tendo em vista os coeficientes de majoração, minoração, aditivos e um modelo de cálculo além do estipulado na norma. Primeiramente são definidos os modelos a serem utilizados: Bielas e tirantes, Atrito-cortante, viga em balanço e finalmente o modelo desenvolvido pelo autor Mounir Khalil El Debs. A continuação, são determinadas as etapas de cálculo e os parâmetros envolvidos em cada etapa, descrevendo também os tipos de consolos considerados. Seguidamente, serão determinadas as fórmulas para o dimensionamento da armadura e verificações de estado limite utilizando as cargas e resistências de projeto definidas. A última etapa apresenta os valores limites de dimensões e esforços. Finalmente será apresentado uma demonstração de aplicação da ferramenta a partir de um exemplo disponível na bibliografia.

Palavras-chave: Parametrização computacional; Pré-moldados; Consolos; Concreto Armado

ABSTRACT

In recent years, Brazil has seen an increase in the use of precast reinforced concrete construction systems, mainly due to the speed of construction. However, there are some inherent difficulties in this construction system. These problems are observed, for example, in beam-column connections made with corbels, and are addressed by several authors, consequently leading to the exploration of various solutions. Therefore, the need arises for a tool that automates and speeds up the design process to advance this research. Thus, this work aims to provide the implementation of a computational algorithm for the design of precast reinforced concrete corbels, complying with Brazilian Technical Standard 9062 – 2017 and considering different types of corbels (according to geometry). This takes into account the coefficients of increase, decrease, additives, and a calculation model beyond that stipulated in the standard. First, the models to be used are defined: Struts and ties, Shear friction, cantilever beam, and finally the model developed by the author Mounir Khalil El Debs. Next, the calculation steps and the parameters involved in each step are determined, also describing the types of corbels considered. Subsequently, the formulas for reinforcement design and limit state verifications using the defined design loads and resistances will be determined. The last step presents the limit values of dimensions and stresses. Finally, a demonstration of how to apply the tool will be presented, using an example available in the bibliography.

Key words: Parametrization computacional; Pre-cast; Corbels; Reinforced concrete

SUMÁRIO

1 INTRODUÇÃO

Embora o sistema tributário, o conservadorismo na construção civil por parte dos agentes atuantes e a instabilidade econômica influenciem na subutilização de pré-moldados El Debs (2017), a industrialização da construção civil em concreto pré-moldado tem ganhado espaço pelo controle e gerenciamento aplicável ao referido sistema construtivo, Reginato (2020).

A construção civil tradicional apresenta vantagens econômicas em obras pequenas, porém a situação é diferente quando trata-se de obras de grande porte, onde o planejamento, a geração de resíduos, melhor aproveitamento de materiais, qualidade dos elementos e velocidade de construção, colocam os pré-moldados como a opção mais preferível, Costa (2009). Tendo em vista isto, as ligações neste tipo de estruturas são pontos críticos, pois são os encarregados da transmissão de cargas, Wissman et al. (2023)

Para El Debs (2017), as ligações em geral são as partes mais importantes em estruturas de concreto pré-moldados, pois são fundamentais para o comportamento da estrutura finalizada. As ligações em pré-moldados são mais deformáveis quando comparadas a ligações monolíticas, Teodoro (2022). A filosofia de projeto para ligações pré-fabricadas aborda tanto os requisitos estruturais quanto o método de construção escolhido e inclusive as práticas de trabalho na fábrica podem determinar como será realizado o projeto das conexões, Elliott (2002). As ligações devem transmitir as cargas, restringir o movimento e promover a estabilidade, PCI (2010, apud Prado et al., 2017). Neste sentido, a NBR 9062 (2017) recomenda a consideração da estabilidade da estrutura, rotações, deformações e/ou deslocamentos possíveis de ocorrer durante a fase de montagem e no uso do elemento.

Os consolos são elementos construtivos utilizados como ligação entre vigas-pilar, viga-viga, laje-viga, laje-pilar e laje-parede, Oliveira (2012). Comumente utilizados quando a questão estética é secundária pois em outros casos é preferível que os consolos estejam contidos dentro da altura da viga, Elliot (2002). São encarregados da transmissão de cargas verticais e horizontais, Leonhardt e Mönnig (1978). O Manual Britânico da BS 8110-1 (1997), considera que os consolos também devem suportar cargas de rolamento. Na figura 1, é possível observar uma ligação viga-pilar através de um consolo.

Figura 1 – Ligação viga-pilar

Fonte: O autor, 2025

Os consolos são geralmente projetados dos pilares servindo de apoio para outros elementos estruturais, Araujo et al. (2016). Efeitos de rotação em vigas podem ocorrer com o avanço da obra e após a terminação da estrutura completa, o que tende a deslocar a resultante da reação do apoio no consolo, Reginato (2020). Se as ligações forem pouco rígidas podem causar grandes momentos positivos no vão da viga o que geraria a necessidade de uma maior taxa de armadura e uma peça mais robusta para resistir os esforços, onerando a obra em questão, Teodoro (2022). Assim, é possível observar que as ligações por meio de consolos são pontos críticos na utilização e na montagem. Porém, o desempenho e o comportamento estrutural desta peça dependem do correto dimensionamento e execução do mesmo. Numa pesquisa comparativa entre modelos de dimensionamento Araújo et al. (2016), mostraram que existe um desvio padrão na avaliação da força de ruina por esmagamento da biela de compressão. Torres (1998), concluiu que taxas de armadura baixas podem ser muito conservadoras, entretanto taxas altas podem, ser contrarias à segurança.

Infelizmente problemas deste tipo transcenderam as pesquisas, como pode ser observado no caso do desabamento de uma estrutura de pré-moldado em 2023, onde segundo a análise de Wissman et al. (2023), um consolo apresentou um tipo de

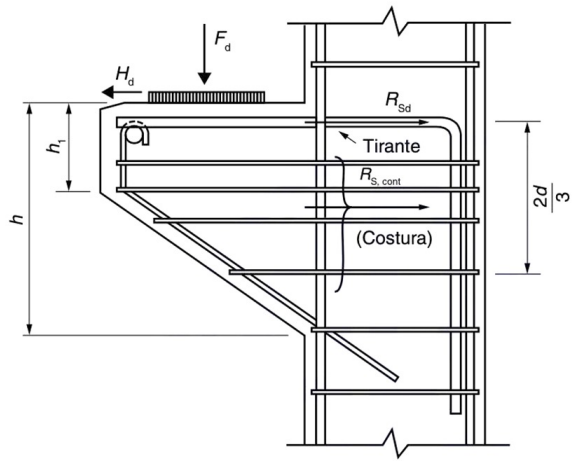
ruptura frágil e concluíram que o desabamento ocorreu por uma execução incorreta da armadura.

1.1 Justificativa

Os consolos são objeto de inúmeros estudos e tendem a intrincar a produção das peças que o possuem devido às altas taxas de armadura normalmente existentes, Oliveira (2012). O seu comportamento é tratado de forma diferente às vigas e apresenta dificuldades devido à grande quantidade de armadura em um espaço pequeno, Costa (2009).

Na figura 2, é possível observar a disposição das armaduras segundo a NBR 6118 (2014), as quais resistem e transmitem a carga vertical F_d e horizontal H_d.

Figura 2 – Disposição de armaduras: Tirante e Costura



Fonte: Adaptado NBR 6118, 2014

As armaduras são dispostas de forma a combater o fluxo de tensões, El Debbs (2017). Segundo o fluxo, a transmissão de cargas verticais e horizontais ocorre com o tirante sendo tracionado e a biela sendo comprimida, Leonhardt e Mönnig (1978). Já as armaduras de costura contribuem para a resistência da força de ruína, Araujo et al. (2016).

Como já foi mencionado as ligações são peças fundamentais para a estabilidade da estrutura. Por tanto, o bom dimensionamento das armaduras e a correta verificação da resistência da peça é de vital importância para a correta transmissão das cargas, a estabilidade e a resistência e da estrutura global.

Mattock (1976), entende que o consolo pode ser tratado como uma peça separada e por tanto o seu dimensionamento se reduz ao calculo da quantidade de armadura necessária para resistir as forças reativas e momentos atuantes.

Ferramentas como modelos de bielas e tirantes e modelos de cisalhamento e atrito, são geralmente necessárias para uma melhor compreensão dos mecanismos de resistência do elemento, Cunha et al. (2014). Existem vários modelos de cálculo que levam a diferentes resultados, Araujo et al. (2016).

Também existem outras abordagens na pesquisa sobre estes elementos. Barreto (2009), concluiu que as fibras de aço aumentam a ductilidade e melhora a resistência à ruptura e finalmente propõe o inicio de uma linha de pesquisa.

Sendo assim, torna-se interessante uma ferramenta que apoie na realização dos cálculos de dimensionamento de acordo com modelos propostos, através da seleção de parâmetros e que apresente os resultados das quantidades de armadura necessária e a verificação de resistência correspondentes ao modelo.

1.2 Objetivos

O objetivo principal é criar uma ferramenta computacional capaz de realizar de maneira prática e automatizada as verificações necessárias para o dimensionamento de consolos de pilares pré-moldados.

Objetivos específicos

1) Implementar os critérios específicos da norma ABNT NBR 9062, o manual do PCI *Design Handbook* e o modelo proposto por El Debs.
2) Fazer as verificações recomendadas em cada modelo proposto.
3) Fornecer um relatório padrão aos usuários com os parâmetros necessários para atendimento dos critérios e os cálculos realizados pelo software.

1.3 Organização do Trabalho

A primeira etapa do trabalho consiste na Revisão Bibliográfica. Nesta etapa será apresentado o comportamento estrutural da peça, as suas características geométricas, as armaduras resistentes e os modelos de calculo propostos na literatura. Também as particularidades e os aspectos considerados nas normas e pelos autores.

A continuação serão apresentados os modelos para a sua implementação computacional, explicando as considerações feitas para as formulações e as adaptações feitas para o desenvolvimento do *software*. Será confeccionado um fluxo de trabalho para a consecução dos resultados no modelo computacional.

Os dados de entrada serão classificados segundo a magnitude física onde o usuário preencherá os campos fazendo a escolha dos materiais, a configuração geométrica, os coeficientes correspondentes ao modelo e as cargas envolvidas.

A continuação será desenvolvido um relatório padrão, onde o usuário poderá conferir as eleições realizadas, as formulações conforme o modelo escolhido e dados adicionais de identificação do projeto.

Finalmente será apresentado um teste do *software* por meio de um problema proposto. Mostrando o funcionamento do software as interfaces e o relatório final dos cálculos feitos.

2 REVISÃO BIBLIOGRÁFICA

Os consolos são balanços muito curtos analisados separadamente, como outros elementos estruturais como vigas e pilares, El Debs (2017). A NBR 6118 (2014) considera como consolos aqueles elementos em balanço cuja distância a, do ponto de aplicação da carga à face do elemento de apoio é menor ou igual á altura útil d, do consolo. Para Russo et al. (2006), os consolos são balanços com relação $a/d \leq 1{,}0$.

Além das várias definições, os consolos podem ser utilizados a diferentes níveis de altura e com diferente formato e projetadas em faces opostas a partir de um mesmo apoio conforme figura 2.

Figura 3 – Consolos projetados em faces diferentes de um pilar

Fonte: O autor, 2025

Para as juntas devem ser indicadas a forma e a posição na superfície horizontal superior do consolo NBR 6118 (2014). A NBR 9062 (2017) explica que as ligações entre os elementos apoiados e o consolo podem ocorrer por meio de juntas a seco, argamassa, concretagem no local, dispositivos metálicos ou apoios elastoméricos entre outros, considerando a ausência impedimento de movimento horizontal na ligação.

Dependendo das características geométricas e a posição dos elementos em volta do consolo estão sujeitos a esforços cortantes, de torção e momento fletor a

partir da carga transmitida pelos elementos apoiados como poderá ser visto a continuação.

2.1 Formatos Geométricos dos Consolos

Existem diversas recomendações sobre a geometria dos consolos, estes variam segundo o modelo proposto e os normativas. O formato mais comumente analisado na literatura é o formato retangular. Neste formato é observado que a região inferior fica isenta de tensões, El Debs (2017), justificando assim a utilização do chanfro para maior optimização de recursos.

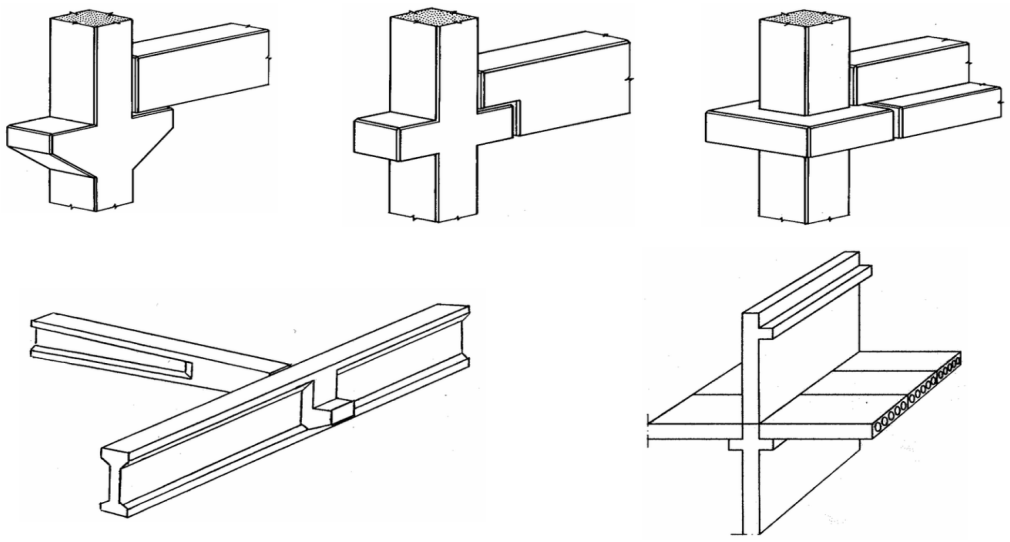
Geralmente, a diferenciação entre consolos e vigas em balanço é dada pelo pela relação entre o ponto de aplicação da carga a e a altura útil d. A ABNT NBR 9062 (2017) classifica como consolos curtos aqueles que cumprem com a condição $0{,}5 < a/d \leq 1{,}0$ e como consolos muito curtos aqueles com relação $0{,}5 \leq a/d$, é utilizado o método de atrito-cizalhamento.

Para Leonhardt e Mönnig (1978) a altura útil deve ser maior que o comprimento l medido desde a interface pilar-consolo e a face externa. O Manual da FIB (2003) considera que o comprimento l pode ser limitado entre 50 e 40 cm e menor ao 70% da altura total h. Já a NBR 9062 (2017), menciona que o comprimento l e a largura b, geralmente igual à largura do pilar, deve ser fixado levando em consideração as folgas para o aparelho de apoio e a ancoragem da armadura e outras disposições construtivas.

Para El Debs (2017) a altura da face frontal pode ser igual à metade da altura total. No caso de utilização de chanfro, o ângulo com a horizontal $\theta \geq 45º$ a e largura não deve superar os 500 mm, Elliott (2002).

Como já foi mencionado, os consolos podem adotar diversos formatos. O manual da FIB (2003), apresenta os seguintes exemplos, sendo os mais comuns e analisados na literatura os de seção retangular e o trapezoidal (ou com chanfro).

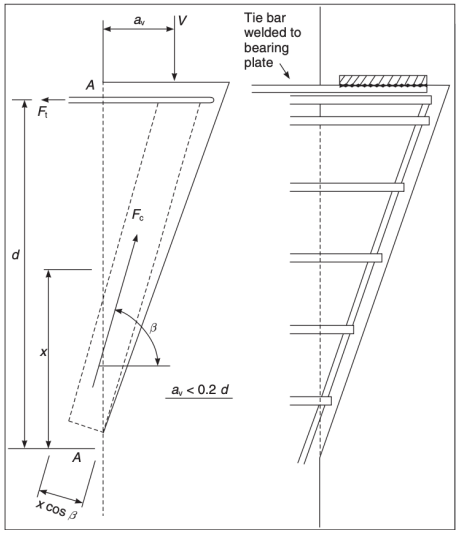
Figura 3 - Exemplos de formatos geométricos e utilização de consolos de concreto pré-moldado



Fonte: Manual FIB, 2003

Também, na literatura podem ser encontrados outros formatos, como o apresentado na figura 4 por Elliott (2002).

Figura 4 - Formato triangular de consolo (Deep Corbel)



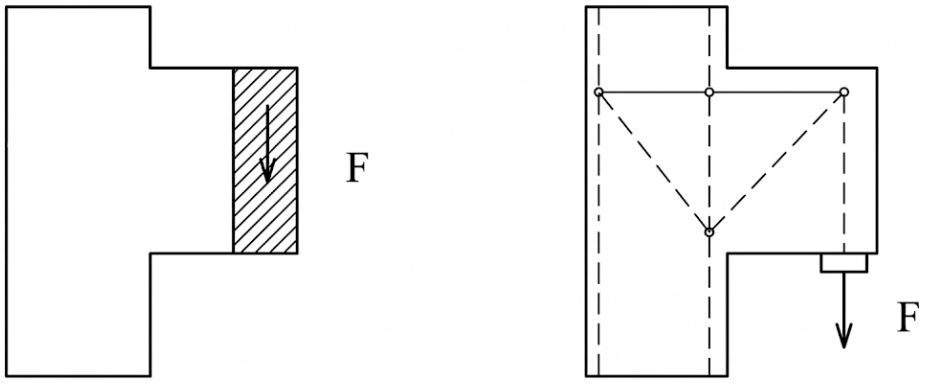
Fonte: Adaptada Elliot, 2002

2.2 Aplicação de Cargas

Segundo Elliott (2002), a carga pode ser aplicada na parte superior do consolo, chamada de aplicação direta. Os consolos são desenhados para resistir as cargas verticais últimas da viga e horizontais da dilatação da viga, Russo et al. (2006).

No entanto, a carga pode ser aplicada ao longo da altura ou na face inferior como observado na figura 5, chamada de aplicação indireta da carga, Torres (1998).

Figura 5 – Aplicação de cargas ao consolo



Fonte: Torres, 2003

Leonhardt e Mönnig (1978), propõem arranjos de armadura para a resistência dos carregamentos indiretos. No primeiro caso a armadura do tirante funciona também como a armadura de suspensão figura 6.a e no segundo caso uma armadura de suspensão inclinada ancorada no pilar suporta parte da carga, figura 6.b.

Figura 6.a – Armaduras para aplicação de carga direta

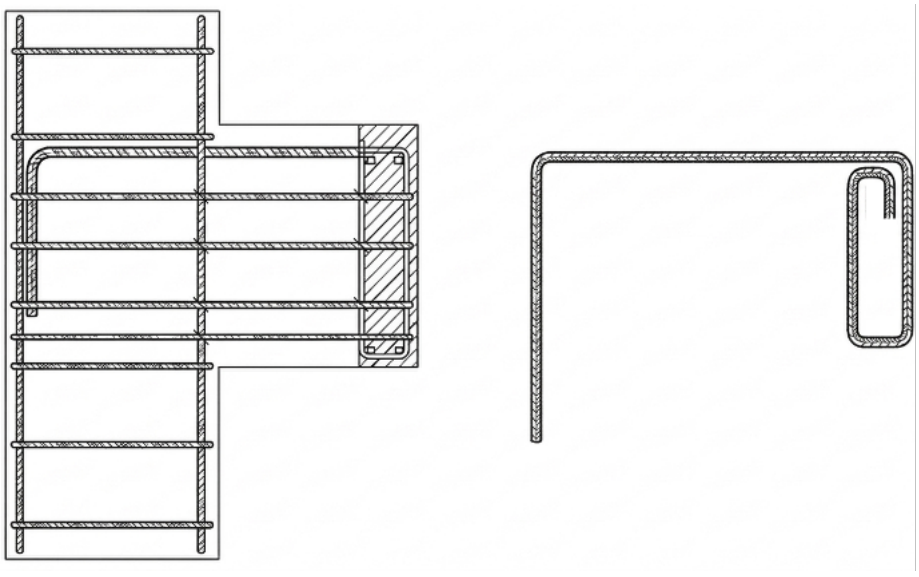
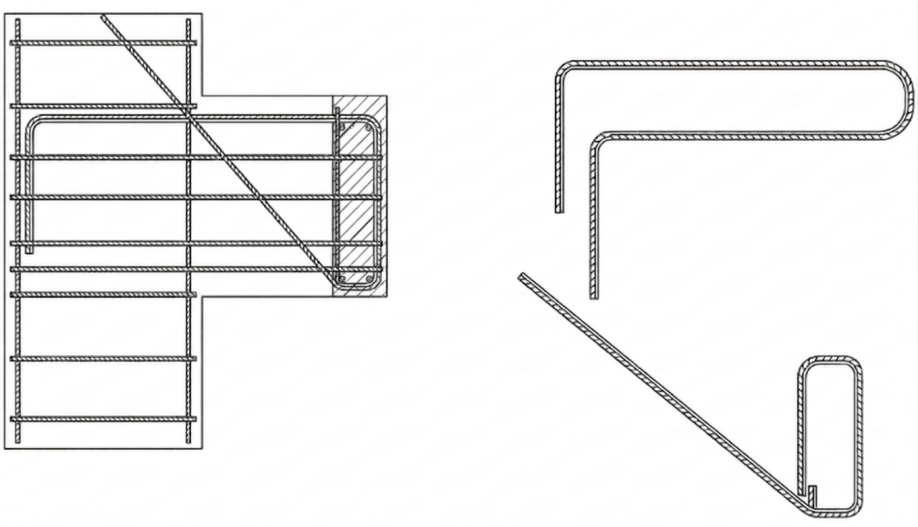


Figura 6.b – Armaduras para aplicação de carga indireta



Fonte: Adaptado Leonhardt e Mönnig vol. 3, 1978

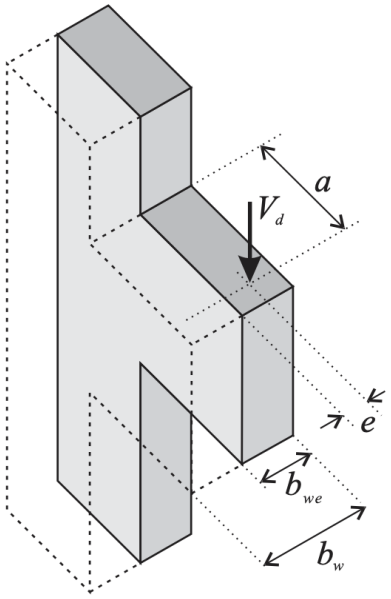
A NBR 9062 (2017) considera que as armaduras de suspensão devem suportar integralmente toda a carga, adotando-se uma tensão de escoamento $fyd \leq 435\ MPa$ e fatores de seguridade segundo o modo de aplicação da carga.

As cargas horizontais são aplicadas região entre o apoio e o elemento apoiado e são fundamentais para o dimensionamento, conforme NBR 6118 (2014). Estas forças podem ser geradas por variações volumétricas, frenagem de pontes rolantes e outras ações eventuais, Reginato (2020).

2.3 Cargas Excêntricas e Torção

A NBR 9062 (2017), considera que a torção gerada por uma força horizontal, transversal ou excêntrica ao longo da largura tem um comportamento estrutural de um modelo biela tirante fora do plano médio do consolo. Reginato (2020) explica que a norma considera um modelo de treliça mais estreita e que isto afeita somente ao modo de verificação da biela de compressão, devido à menor largura da treliça.

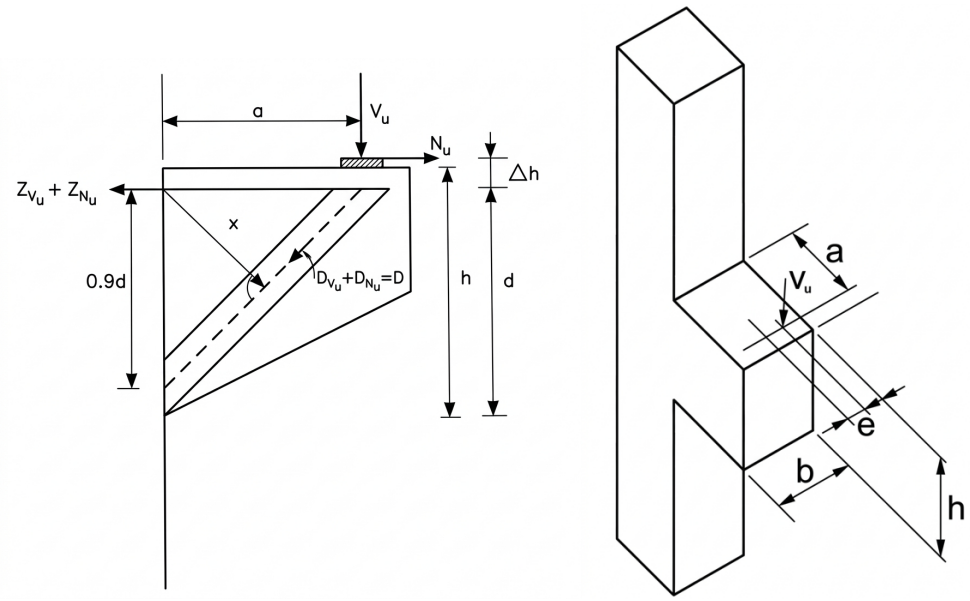
Figura 7 – Modelo de treliça devido à carga excêntrica conforme a NBR 9062



Fonte: Adaptado Reginato, 2020

Solanki e Sabnis (1987), propõem um modelo onde a área de armadura é determinada a partir do momento causado pelo carregamento vertical na treliça formada (figura 8) com o braço de alavanca sendo obtida pela formação do triângulo retângulo entre o valor da excentricidade, o ponto de aplicação de carga e o braço.

Figura 8 – Modelo de treliça proposto devido à carga excêntrica



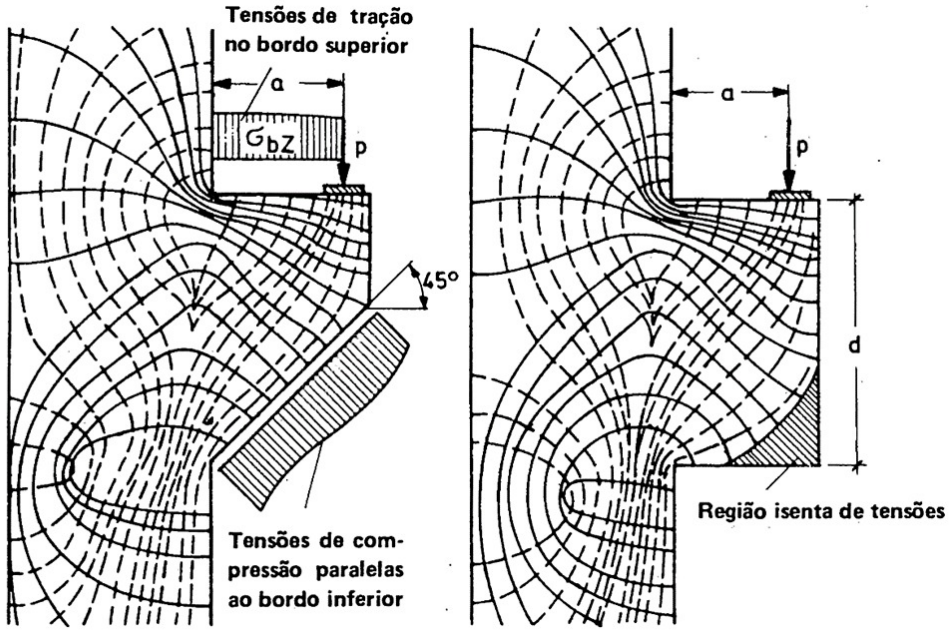
Fonte: Adaptado Solanki e Sabnis, 1987

2.4 Comportamento estrutural

A tensão de compressão parte do ponto de aplicação da carga até a base do consolo e no caso do consolo com chanfro, a trajetória é paralela à face inferior, El Debs (2017). Devido a este comportamento é fundamental a verificação da resistência do concreto para evitar uma ruptura brusca, Carvalho et al. (2016). Isto último torna-se uma tarefa dificultosa na hora de determinar um nível de tensão para essa região uma vez que existe uma diversidade de geometrias possíveis para a mesma, embora o cruzamento entre as tensões indique um formato de garrafa para a biela, Canha et al. (2014), o que será observado nos modelos de dimensionamento disponíveis.

As isostáticas de tração são ligeiramente inclinadas e são constantes ao longo do comprimento a, El Debs (2017). Também, da análise feita por Franz e Niedenhoff (1963, apud Leonhardt e Mönnig, 1978) é possível observar uma distribuição de tensões de tração ao longo da altura total do consolo que se concentra mais na região superior até a metade da altura total, conforme a figura 9.

Figura 9 – Distribuição de tensões em consolos curtos. Linhas contínuas (tração). Linhas tracejadas (compressão).

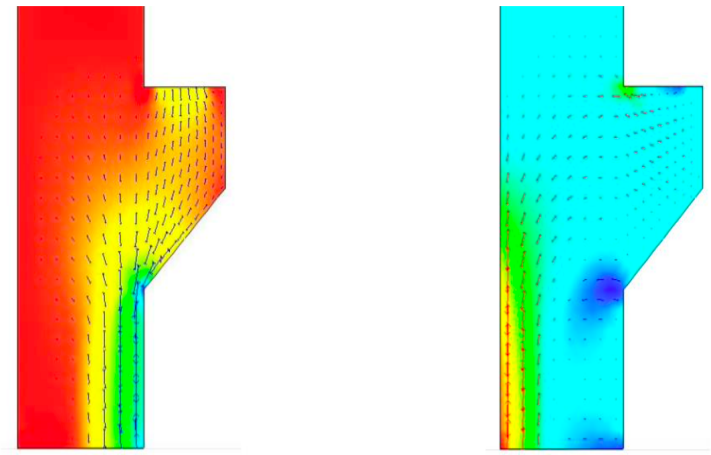


Fonte: Adaptado Leonhardt e Mönnig vol. 2, 1978

Se bem os consolos mais analisados na literatura são os consolos curtos, é possível observar que o comportamento se estende a consolos considerados como muito curtos. Carvalho et al. (2016), mostram que a trajetória de tensões, para consolos classificados como muito curtos pela NBR 9062 (2017), é praticamente a

mesma conforme a figura 10, com as tensões de compressão atravessando a interface consolo-pilar, e com a concentração de tensões de tração perto do ponto de aplicação da carga.

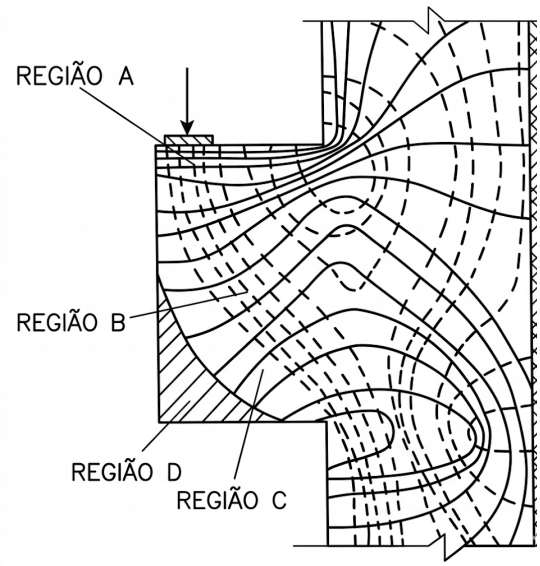
Figura 10 – Distribuição de tensões em consolos muito curtos.



Fonte: Carvalho et al., 2016

A partir dos estudos de Franz e Niedenhoff em 1963, Torres (1998) classifica as regiões de tensões conforme a figura 11, sugerindo que na região A seja colocada a armadura principal e observa que na região B as tensões de compressão são quase constantes, surgindo assim a biela comprimida. Na região C ocorre uma mudança de direção na trajetória das tensões de compressão (tracejadas), provocando pequenas tensões de tração. Finalmente a região D, está reservada consolos com cantos retos e está isenta de tensões.

Figura 11 - Regiões de tensão



Fonte: Adaptado Torres, 1998

2.5 Armadura Principal (Tirantes)

É a parte principal do sistema de equilíbrio interno dos consolos. A trajetória quase horizontal trajetória da tração na zona superior indica a necessidade de colocação de armadura, sendo esta a armadura principal no modelo de bielas e tirantes, Canha et al. (2014).

2.6 Armadura Secundária (Costura)

São camadas de armadura posicionadas sob a armadura principal. O PCI Design Handbook (2010) recomenda que sejam distribuídas em 2/3 da altura útil d. Russo et al. (2006), mencionam que as camadas finais podem não escoar. No entanto, modelos de cálculo como o proposto por Mattock et al. (1976), consideram que estas armaduras também escoam. Alguns modelos consideram a armadura de costura como uma fração do tirante, outras consideram um novo sistema de biela e tirante, Araújo et al. (2016).

2.7 Armaduras Verticais (Estribos)

A norma NBR 9062 (2017) recomenda a utilização destas armadura. Já o PCI Design Handbook (2010), não as considera no dimensionamento das armaduras. Contribuem na distribuição de tensões e na ductilidade da peça, porém não aumenta a resistência dos consolos, Canha et al. (2014). Leonhardt e Mönnig (1978), consideram que estas armaduras são inúteis para a transmissão de carga ao consolo e serve só para enrijecer a armadura.

2.8 Bielas de concreto

Hwang et al. (2000, apud Russo, 2006), consideram que o eixo da biela coincide com a direção da compressão no consolo. As armaduras de costura combatem a fissuração na biela Costa (2009).

2.9 Mecanismos de ruptura

A ruptura do consolo pode ocorrer por diferentes causas, desde erros no detalhamento da armadura até erros de análise estrutural e isso consequentemente gera diferentes tipos de ruptura. O modo de ruptura depende essencialmente da quantidade de armadura utilizada, resistência à compressão do concreto e da razão a/d. Os mecanismos citados a continuação são os que governam os modelos de cálculo, El Debs (2017):

1. Deformação excessiva da armadura levando ao esmagamento do concreto contra à face do pilar
2. Esmagamento do concreto a partir do ponto de aplicação da carga até a face do pilar
3. Escorregamento na interface consolo-pilar causado por corte direto

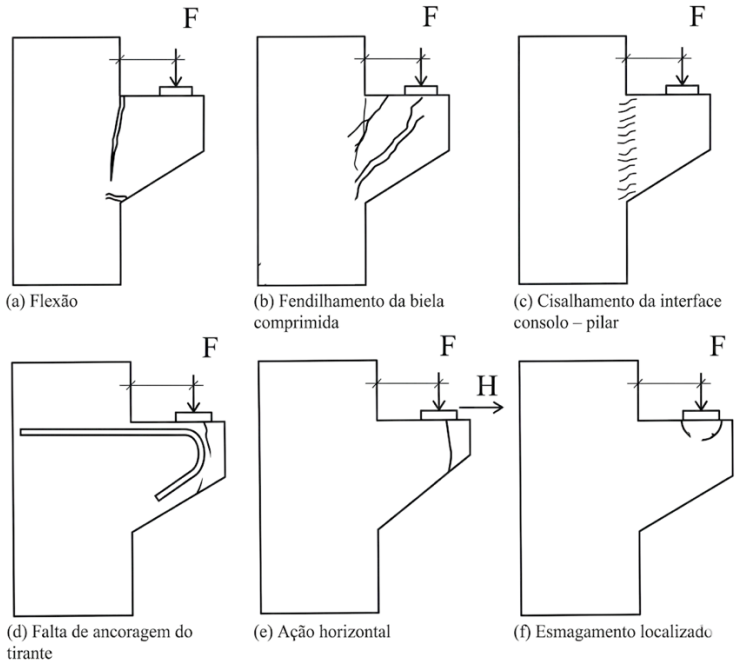
Park e Paulay (1983, apud Torres, 1998), observam que a ruína por flexão (1) gera grandes fissuras na interface consolo-pilar, a ruína por fendilhamento da biela (2) inicia com fissuras de flexão seguida de fissuras que se propagam quase linearmente desde o ponto de aplicação da carga até o canto inferior do consolo, a ruína por cisalhamento (3) produz fissuras inclinadas na interface-consolo pilar.

Canha et al. (2014), mencionam que a falha por flexão ocorre depois do escoamento da armadura principal e é causado pela baixa taxa de armadura e altos valores da relação a/d. Consolos muito curtos geralmente rompem por cisalhamento, embora também possa acontecer para consolos curtos, por tanto é recomendável a o bom dimensionamento e detalhamento de armaduras de costura para prevenir estes acontecimentos, Machado e Pimenta, (2000).

Segundo Russo et al. (2016), a utilização de armadura horizontal gera uma tendência dos modos de falha à convergência de um tipo de falha denominado como falha por cisalhamento da viga.

Araújo et al. (2017), ainda mencionam os seguintes tipos de falha: por falta de ancoragem do tirante, por ação horizontal e por esmagamento localizado.

Figura 12 – Mecanismos de ruptura



Fonte: Araújo, 2017

3 MODELOS DE DIMENSIONAMENTO

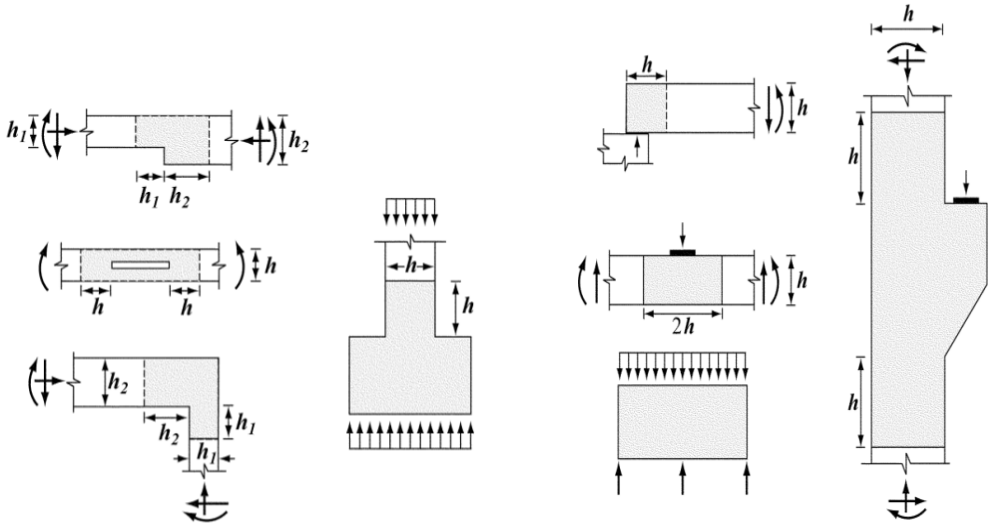
Nesta seção serão apresentados os modelos implementados na ferramenta computacional e as equações que sustentam os mesmos. A ferramenta é interativa e os parâmetros podem ser mudados para uma maior manipulação dos resultados. Por motivo de simplificação e o interesse de criar uma ferramenta que possibilite a manipulação dos parâmetros para fins de experimentação e desenvolvimento de pesquisas, foram feitas algumas considerações sobre os modelos que serão elencados a continuação e pelo mesmo motivo a ferramenta se centrará no calculo das armaduras e verificação das bielas:

1. A aplicação das recomendações sobre a geometria, relatadas no capítulo 2 submetidas à consideração do usuário.
2. Os fatores de segurança aplicados às cargas e resistência dos materiais que não formam parte das equações de cálculo também são deixados à consideração do usuário.
3. O software realiza o cálculo das armaduras principais e secundarias. As armaduras verticais não formam parte do mecanismo resistente dos modelos propostos.
4. A verificação dos nós no caso do modelo de bielas e tirantes fica fora do escopo, posto que nem todos os modelos implementados apresentam este passo.
5. O sistema de unidades depende do prescrito no modelo proposto.
   Outras particularidades são descritas na apresentação de cada modelo.

3.1 Bielas e tirantes

A utilização do modelo de bielas e tirantes é permitido em regiões D, onde as hipóteses da seção plana não se aplica NBR 6118 (2014). As regiões D são aquelas partes da estrutura separadas a uma determinada distância de descontinuidades de forças ou geométricas ACI 318 05, (2005). Dita delimitação pode ser realizada em função da altura do elemento medidas a partir da descontinuidade, Canha et al. (2014). Nestas regiões não são válidas as hipóteses de Bernoulli e o fluxo de tensões não é constante nem uniforme contrario ao Princípio de Saint-Venant, Reginato (2020).

Figura 13 – Regiões B e D. Regiões D em cinza.



Fonte: ACI 318 05, 2005.

O modelo também conhecido como modelo de treliça, consiste em idealizar o fluxo de tensões da estrutura, substituindo os fluxos por elementos tracionados e comprimidos Barreto (2009). A partir de uma analise linear da peça a ser dimensionada, os fluxos de tração são substituídos por elementos denominados tirantes e os fluxos de compressão por elementos denominados bielas, Reginato (2020).

El Debs (2017), observa que na maioria dos modelos de bielas e tirantes não é comum a realização de verificação da resistência dos nodos da treliça. Porém, a NBR 6118 (2014), define como zona nodal um volume determinado de concreto ao redor dos nós e deve ser verificada a resistência e explica que nestes pontos da treliça são aplicadas as forças externas e reações de apoio. Tanto a ACI 318 05, (2005) como a NBR 6118 (2014), apresentam os seguintes fatores ponderados para a verificação de zonas nodais e bielas de compressão, como pode ser verificado na Tabela 1.

Tabela 1 – Fatores de ponderação para zonas nodais.

	NBR 6118	ACI 318 05
fc'= α(1)*β*fcd (2)	β	
Nós com 1 tirante	0,72	0,80
Nós com mais de 1 tirante	0,60	0,60
Zonas somente comprimidas/ Zonas estruturais ou de apoio	0,85	1,00

1. No caso da NBR 6118 o valor do $\alpha = 1 - \frac{fck}{250}$. Já para a ACI 318 05 $\alpha = 0{,}85$.

2. Os símbolos das variáveis fc', α e β foram padronizados para uma melhor explicação.

Fonte: Adaptado NBR 6118, 2014 e ACI 318 05 (2005)

3.1.1 NBR 9062 - Biela e Tirante

Esta norma prescreve a utilização do modelo biela e tirante para consolos curtos definidos como consolos onde se cumpre a condição da seguinte condição:

$$0{,}5 < \frac{a}{d} \leq 1{,}0 \quad (1)$$

3.1.1.1 Armaduras:

A armadura principal ou do tirante é calculada pelas equações 2 e 3, onde é possível notar que a armadura do tirante é calculada em função da contribuição da carga vertical e horizontal. Já a armadura de costura é calculada pela equação 4, como uma parcela da armadura resistente à carga vertical.

$$A_s = A_v + \frac{H}{fy} \quad (2)$$

$$A_{sv} = \left(0{,}1 + \frac{a}{d}\right) * \frac{V}{fy} \quad (3)$$

$$A_{sc} = 0{,}4 * \frac{A_{sv}}{d} \quad (4)$$

Onde fy a resistência ao escoamento escolhida.

3.1.1.2 Verificação das bielas:

A norma não prescreve um método de cálculo para a tensão de compressão atuante na biela de compressão. Por tanto o usuário deverá realizar um cálculo separadamente para a determinação da compressão atuante na biela, e compara-las com os limites impostos pela norma.

Para a tensão de compressão na biela inclinada não podem ser ultrapassados os seguintes limites em função da aplicação da carga.

Tabela 2 – Limites para resistência da biela segundo a aplicação da carga

Aplicação da carga	Limite de Resistência da Biela
Direta	fc
Indireta	$0{,}85 * fc$

Fonte: NBR 9062, 2017

Onde fc' é calculada conforme a Tabela 3 segundo a situação da biela analisada:

Tabela 3 – Fatores de forma para calculo da resistência da biela

$fc' = \alpha * \beta * fc$	β
Bielas Atravessadas por 1 tirante	0,72
Bielas Atravessadas por mais de 1 tirante	0,60
Bielas Prismáticas	0,85

com $\alpha = 1 - \frac{fck}{250}$, fc' é a resistência à compressão da biela e fc a resistência à compressão do concreto escolhido

Fonte: NBR 6118, 2014

3.1.1.3 Cargas Horizontais:

Na ausência de impedimento ao movimento horizontal a norma permite que a ação horizontal seja estimada em função da carga vertical, conforme a Tabela 4. Isto aplica para todos os modelos de cálculo para consolos disponíveis na NBR 9062.

Tabela 4 – Fatores ponderativos para o calculo da carga horizontal segundo o tipo de apoio ou ligação

Tipo de apoio ou Ligação	H
Juntas a seco	0,8*V
Assentado com argamassa	0,5*V
Elastômero	0,16*V
Com plástico politetrafluoretileno PTFE	0,08*V
Chapas metálicas não soldadas	0,25*V
Apoios com chapas metálicas	0,4*V
Concretagem no local, Ligação por meio de Solda ou Apoio com graute	C *V
Outros valores	C *V

Fonte: NBR 9062, 2017

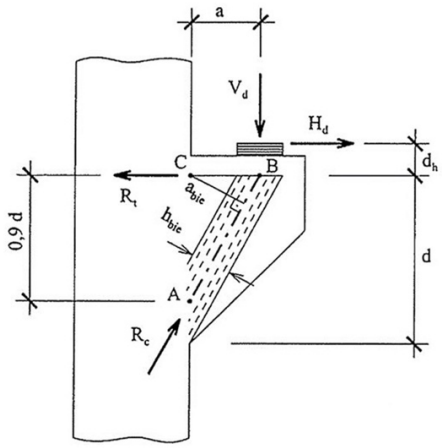
Onde C é um valor fornecido pelo usuário. A adoção deste valor deve ser devidamente justificado. A carga vertical V é considerada como fatorada.

3.1.2 El Debs - Biela e Tirante

Para consolos curtos com relação a/d conforme a equação 5 o autor propõe um modelo de bielas e tirantes, com a treliça formada segundo a Figura 14.

$$0{,}5 < \frac{a}{d} \leq 1{,}0 \qquad (5)$$

Figura 14 – Esquema do sistema de bielas e tirantes proposto por El Debs



Fonte: Adaptado El Debs, 2017

3.1.2.1 Armaduras:

A partir do momento em torno ao Ponto A obtém-se a equação para o dimensionamento da armadura principal. O autor considera que o 50% do momento da carga vertical V é suportado pela armadura de costura conforme a equação 7.

$$A_s = \frac{V * a}{0{,}9 * fy * d} + 1{,}2 * \frac{H}{fy} \tag{6}$$

$$A_{sc} = 0{,}5 * \frac{V * a}{0{,}9 * fy * d} \tag{7}$$

3.1.2.2 Verificação das bielas:

El Debs considera que a tensão atuante pode ser calculada em função da tensão de cisalhamento segundo a equação 8. Já para a tensão limite o autor propõe a equação 9, deduzida a partir do momento ao redor do ponto C.

$$\tau_w = \frac{V}{b * d} \tag{8}$$

$$\tau_{wu} = \frac{0{,}18 * \beta * fc}{\sqrt{0{,}9^2 + \left(\frac{a}{d}\right)^2}} \tag{9}$$

Com β em função da aplicação da carga conforme a NBR 9062, veja-se Tabela 2

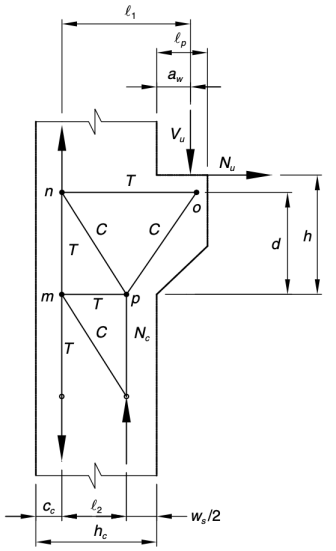
3.1.3 PCI – Bielas e Tirantes (*Strut and Tie method*)

Neste caso, o manual adota um sistema de treliças conforme a figura 15. As posições dos nós, são como segue:

O nó m está localizado na interseção do reforço de tensão da coluna e do tirante na fibra inferior da consolo. O nó n está localizado na interseção da barra de tensão da coluna e o tirante de tração superior do consolo. O nó o está localizado na intersecção da resultante das reações aplicadas (Vu e Nu) e do tirante de tração superior do consolo. O nó p está localizado na intersecção da coluna longitudinal de compressão e do tirante de tração inferior. A largura da biela de compressão ws, que é o fator determinante para localizar o nó p, é determinada pelo equilíbrio de momentos em torno de m.

Para a geração da treliça a ferramenta utiliza um sistema matricial como poderá ser observado no apêndice. A efeitos de simplificação, a determinação da posição dos nós e a largura da biela será realizado num cálculo separado pelo usuário. Por tanto, o mesmo terá de entrar com a posição x e y dos nós no sistema de referência adotado e em seguida definir as barras da treliça.

Figura 15 – Esquema do sistema de bielas e tirantes proposto pelo PCI *Design Handbook*



Fonte: Adaptado PCI Design Handbook, 2010

O usuário deverá definir os apoios considerados para o sistema analisado e as forças externas atuantes.

A continuação são calculadas as forças atuantes no sistema de treliças, definindo o sentido de ação das mesmas em cada barra.

3.1.3.1 Armaduras:

Uma vez calculadas as forças atuantes nas barras o algoritmo realiza o calculo da área de armadura para a força calculada.

$$A_s = \frac{F}{\emptyset * fy} \tag{10}$$

Onde $\emptyset = 0{,}75$, é o fator de minorativo de segurança

3.1.3.2 Verificação das bielas:

Para a verificação da biela de compressão o manual prescreve os seguintes parâmetros de cálculo conforme a ACI 318-05:

Tabela 5 – Fatores do tipo de concreto do consolo considerado

Tipo de concreto	λ
Concreto Normal	1,00
Leve com areia	0,85
Outro tipo de concreto leve	0,75

Fonte: ACI 318-05, 2005

Tabela 6 – Fatores de forma da biela analisada

Fator de forma	βs
Seção transversal uniforme	1,00
Garrafa com armadura	0,75
Garrafa sem armadura	0,60*λ
Biela em zona de tensão	0,40
Outro	0,60

Fonte: ACI 318-05, 2005

O algoritmo a continuação calcula a resistência ultima do concreto e a área da biela conforme segue:

$$fcu = 0{,}85 * \beta s * fc \quad (11)$$

$$A_{c\,s} = b * b_s \quad (12)$$

Onde fcu é a resistência à compressão da biela, $A_{c\,s}$ é área da biela e b_s é a largura da biela.

A continuação é calculada resistência da biela e comparada com o valor obtido no calculo do algoritmo para a barra comprimida, onde a deve ser cumprida a condição da equação 14:

$$F_R = fcu * A_{c\,s} * \emptyset \quad (13)$$

$$Bc \leq F_R \quad (14)$$

Caso a condição não seja atendida o relatório apresentado indicará que a biela não resiste. Neste caso, Bc é a força de compressão atuante na treliça e F_R a resistência da barra.

3.2 Atrito-Cisalhamento

Este modelo é pressupõe uma fissura ao longo do plano de ligação consolo-pilar o que ativa um mecanismo de transmissão de esforços tangenciais pela superfície irregular e a armadura distribuída atravessando dito plano, Machado e Pimenta (2000). As armaduras contribuem à resistência ao corte atravessando a fissura pressuposta gerando uma força normal entre as partes e um efeito de pino Costa (2009).

A biela segue um fluxo mais curto e menos inclinada, devido à baixa relação a/d, gerando uma fissura na interface consolo-pilar e assim transferindo as tensões cisalhantes pelo engrenamento dos agregados e, consequentemente devido ao deslizamento, as armaduras dificultam o afastamento entre as faces, Carvalho et al. (2016).

3.2.1 NBR 9062 – Atrito-Cisalhamento

A Norma Brasileira prescreve a utilização do modelo atrito-cisalhamento para consolos muito curtos, classificados segundo a condição a seguir.

$$0{,}5 \leq \frac{a}{d} \quad (15)$$

3.2.1.1 Armaduras:

Para a armadura do tirante principal, tem-se as equações 16 e 17. Já para a armadura de costura, a equação 18 considera-se o 50% da armadura resistente à carga vertical diferente do 40 % considerado para a mesma armadura no caso de consolos curtos.

$$A_s = A_v + \frac{H}{fy} \tag{16}$$

$$A_{sv} = (0{,}8) * \frac{V}{fy * \mu} \tag{17}$$

Tabela 7 – Fatores ponderativos de lançamento do concreto para interface consolo-pilar

Condição da interface "fissurada"	$\boldsymbol{\mu}$
Concreto lançado monolíticamente	1,40
Superfície rugosa	1,00
Superficie lisa (concreto endurecido)	0,60

Fonte: NBR 9062, 2017

$$A_{sc} = 0{,}5 * \frac{A_{sv}}{d} \tag{18}$$

Salienta-se que a norma permite a consideração do efeito de engrenamento dos agregados desde que a interface de ruptura esteja atravessada por barras de aço, porém é desconsiderado o efeito favorável (compressão da perpendicular à interface) da ação horizontal.

3.2.1.2 Verificação da biela:

As condições de compressão diagonal para este tipo de consolos é feita por meio da tensão de cisalhamento, equação 8. Neste caso, a tensão última também é limitada conforme as equações 19 e 20. A norma não prescreve um modelo de cálculo para a tensão de cisalhamento atuante, neste caso o algoritmo abre um campo em branco para que o usuário forneça o valor calculado pelo modelo escolhido.

$$\tau_{wu} = 3{,}0 + 0{,}9 * \rho * fy \leq 0{,}27 * \left(1 - \frac{fck}{250}\right) * fc \tag{19}$$

$$\tau_{wu} \leq 8\ MPa \tag{20}$$

$$\rho = \frac{A_s}{b * d} \tag{21}$$

$$\tau_{wd} \leq \tau_{wu} \tag{22}$$

Com $fy \leq 435\ MPa$. Onde ρ é a taxa de armadura na interface consolo-pilar e τ_{wd} é a tensão calculada pelo usuário.

No caso em que os limites da tensão de cisalhamento máximo τ_{wu}, sejam ultrapassados o algoritmo assume o valor mínimo entre as duas limitantes como sendo a tensão de cisalhamento máximo τ_{wu}.

3.2.1.3 Cargas Horizontais:

Como já foi mencionado, as cargas horizontais podem ser estimadas conforme a Tabela 4, segundo a NBR 9062.

3.2.2 El Debs – Atrito-cisalhamento

Este modelo é proposto pelo autor para consolos muito curtos com relação a/d segundo a equação 15. A armadura do tirante principal é dada pela equação 23. E a armadura de costura dada pela equação 24.

$$A_s = \frac{1}{fy} * \left[\left(\frac{0{,}8 * V}{\mu}\right) + H\right] \tag{23}$$

$$A_{sc} = 0{,}5 * \frac{1}{fy} * \left(\frac{V * a}{0{,}9 * d}\right) \tag{24}$$

3.2.2.1 Verificação da biela:

Segundo o autor, a tensão atuante também pode ser calculado como no modelo proposto para o caso de bielas e tirantes, conforme a equação 8. Para a verificação da resistência da biela, o autor utiliza o prescrito na NBR 9062 conforme as equações 19, 20, 21 e 22.

Neste caso o algoritmo segue a mesma lógica aplicada no caso do modelo proposto pela NBR 9062, quando a tensão de cisalhamento máximo é ultrapassado assume o valor mínimo entre as duas limitantes como sendo a tensão de cisalhamento máximo τ_{wu}.

3.2.3 PCI – Viga em Balanço (Cantiliver beam)

O manual do PCI na seção 5, propõe um método baseado no modelo de bielas e tirantes e do atrito cisalhamento, conforme é proposto pela ACI 318-05, Reginato (2020). O método está disponível para as condições descritas nas equações 26 e 27. As cargas são consideradas já fatoradas e em kilolibras *(kip)*, as dimensões em polegadas (*in*) e as resistências dos materiais em kilolibras por polegadas ao quadrado (*ksi*).

$$a = \frac{3}{4} * l \tag{25}$$

$$\frac{a}{d} \leq 1{,}0 \tag{26}$$

$$H \leq V \tag{27}$$

3.2.3.1 Armaduras:

A armadura principal é o dado pelo maior entre A_{s1} e A_{s2} conforme equação 31, sendo elas calculadas pelas equações 28 e 30. A armadura mínima do tirante é calculada conforme a equação 32 e comparada com o valor obtido na equação 31.

Já para a armadura de costura o manual desconsidera a parcela resistente à carga horizontal da área do tirante. Neste caso o algoritmo imprime uma mensagem de advertência ao usuário informando o que prescreve o modelo (equação 35) e a continuação a área de armadura de costura calculada pela equação 36.

$$A_{s1} = \frac{1}{\emptyset * fy}\left[\left(V * \frac{a}{d}\right) + \left(H * \frac{h}{d}\right)\right] \tag{28}$$

Neste caso, o fator λ do tipo de concreto depende do estabelecido na Tabela 5.

Tabela 8 – Fatores ponderativos de lançamento do concreto para interface consolo-pilar

Condição da interface "fissurada"	μ	μ_e máximo
Concretado monolíticamente	1,4λ	3,4
Superfície rugosa	1,0λ	2,9
Superficie lisa (concreto endurecido)	0,6λ	Não aplicável
Concreto contra acero	0,7λ	Não aplicável

Fonte: PCI Design Handbook, 2010

$$\mu_e = \frac{\emptyset * 1000 * \lambda * Acr * \mu}{V} \leq \mu_e \, máximo \tag{29}$$

$$A_{s2} = \frac{1}{\emptyset * fy}\left[\left(\frac{2 * V}{3 * \mu_e}\right) + H\right] \tag{30}$$

$$A_{s_t} = \max\ [A_{s1}, A_{s2}] \tag{31}$$

$$A_{s\_min} = \ 0{,}04 * b * d * \left[\frac{fc}{fy}\right] \tag{32}$$

$$A_{s\_min} \leq A_{s_t} \tag{33}$$

Onde μ_e e μ, são os coeficientes de fricção recomendados no manual.

No caso em que $A_{s\_min} > A_{st}$ o algoritmo abre uma janela ao usuário para a escolha entre A_{s_min} e A_{s_t}.

$$A_n = \frac{H}{\emptyset * fy} \tag{34}$$

$$A_{sc} < \ 0{,}5 * (A_s - A_n) \tag{35}$$

$$A_{sc} = 0{,}5 * (A_s - A_n) \tag{36}$$

3.2.3.2 Carga de cisalhamento máximo:

A verificação contra a ruína por cisalhamento vem dada em função da carga vertical máxima conforme a Tabela 9.

Tabela 9 – Fatores ponderativos para interface consolo-pilar para verificação da carga vertical máxima

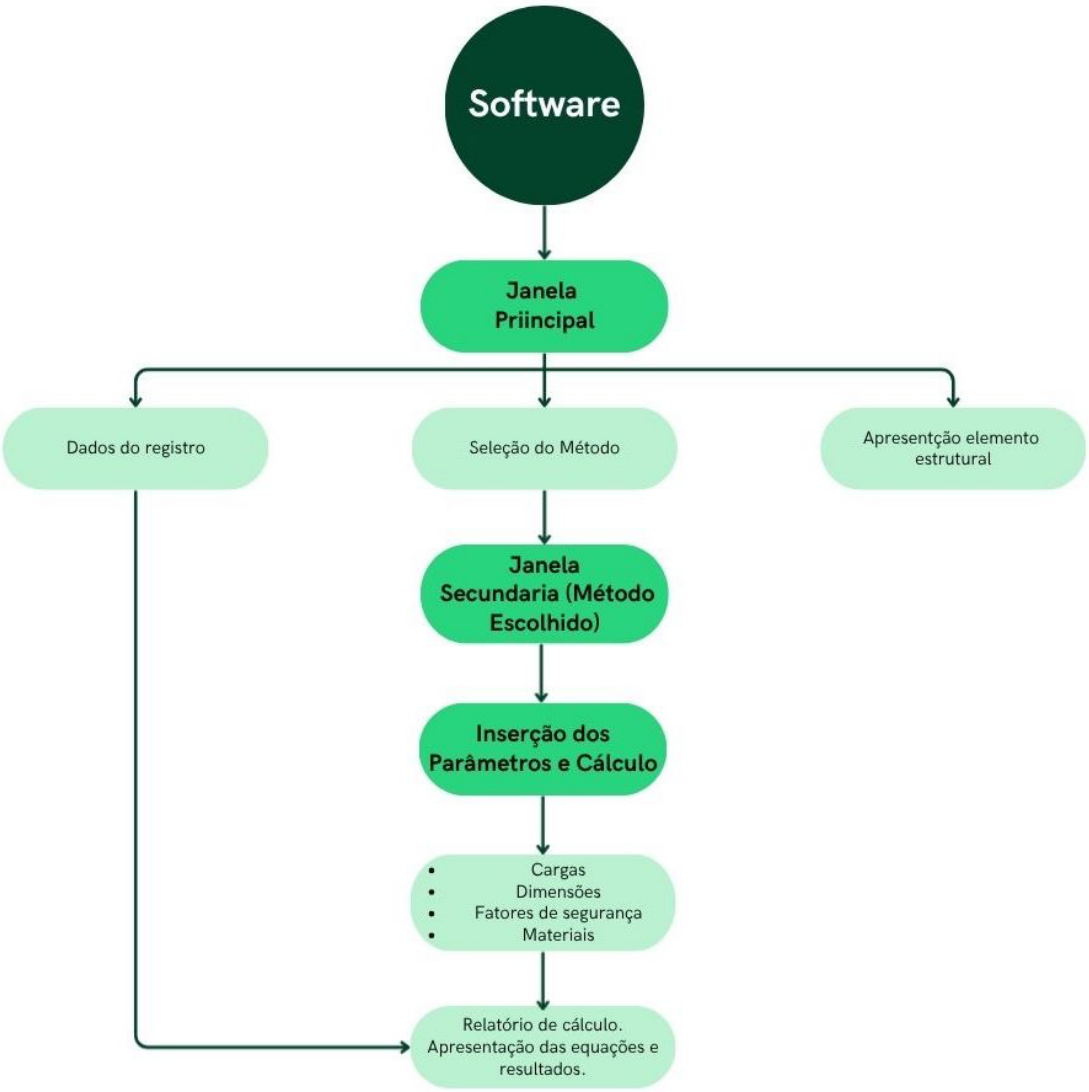
Condição da interface fissurada	**μe máximo**
Concretado monolíticamente	$V_{max} = \frac{\emptyset * 1000 * \lambda * Acr}{1000}$
Superfície rugosa	$V_{max} = \frac{\emptyset * 1000 * \lambda * Acr}{1000}$
Superficie lisa (concreto endurecido)	$V_{max} = \frac{\emptyset * 800 * \lambda * Acr}{1000}$
Concreto contra acero	$V_{max} = \frac{\emptyset * 800 * \lambda * Acr}{1000}$

Fonte: PCI Design Handbook, 2010

4 FERRAMENTA COMPUTACIONAL

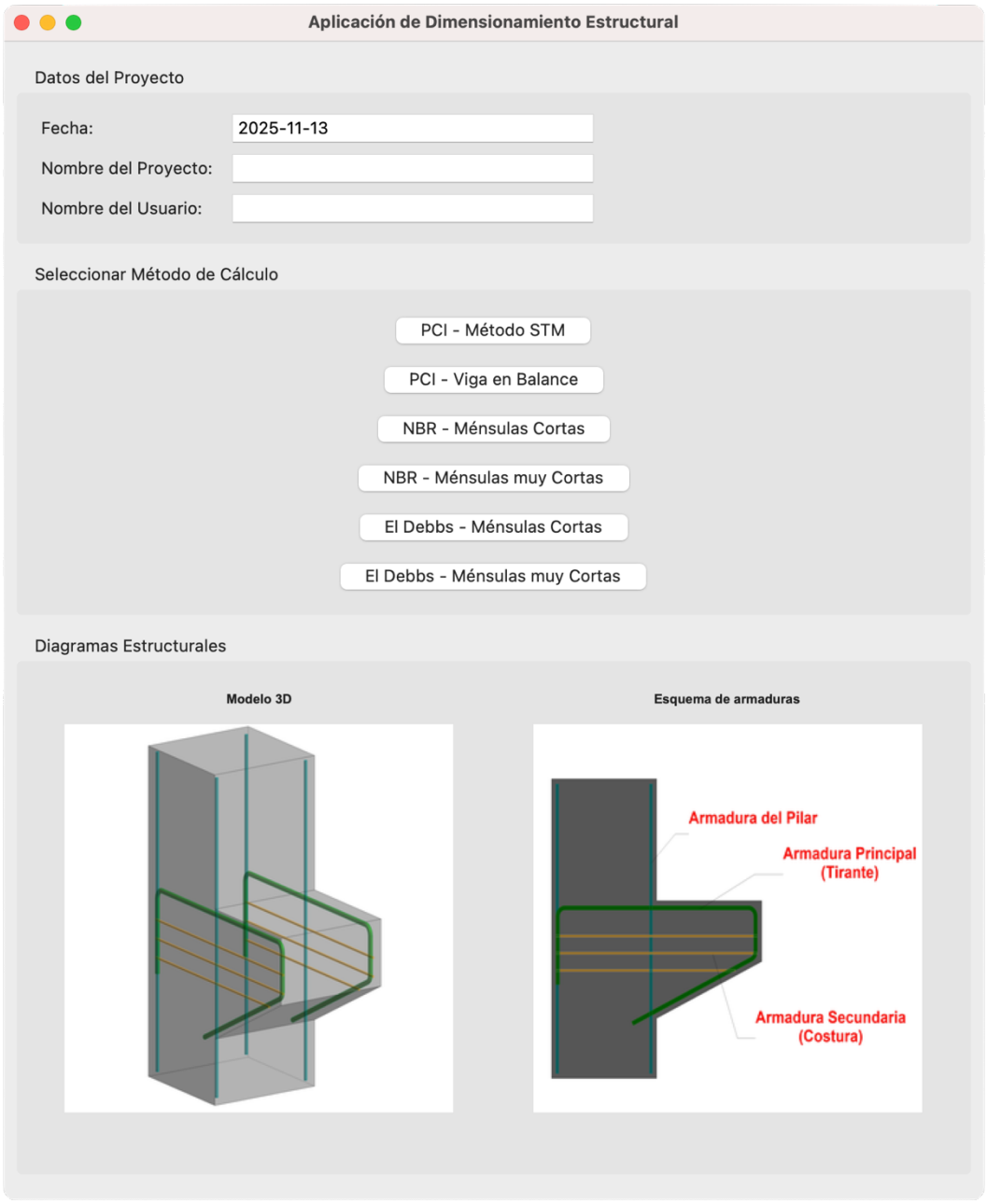
A ferramenta desenvolvida na linguagem de programação Python segue o esquema apresentado na Figura 15. A janela principal da Figura 16, disponibiliza campos de registro para o usuário, apresenta os modelos de cálculo disponíveis e o modelo estrutural considerado para o calculo matemático.

Figura 15 – Fluxograma de funcionamento da ferramenta



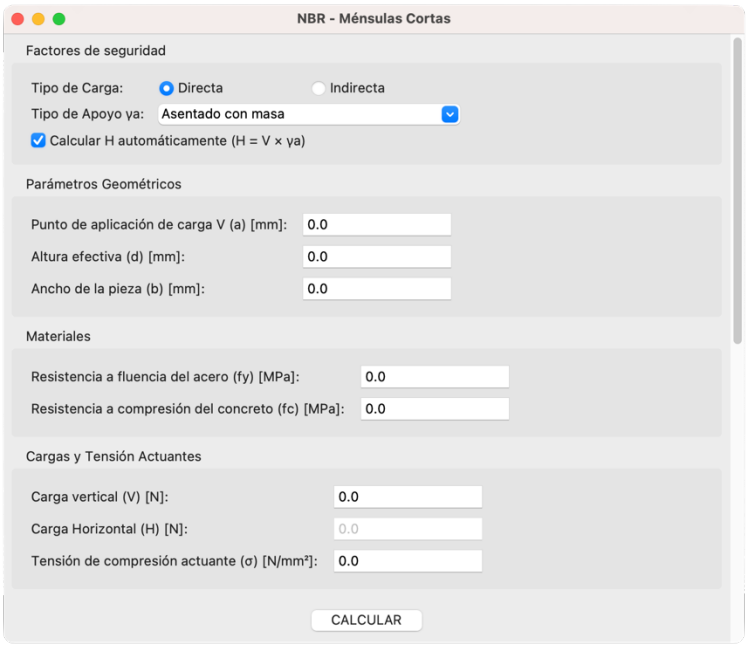
Fonte: O autor, 2025

Figura 16 – Janela Principal do Software



A continuação o usuário deverá selecionar um método. Neste ponto uma janela secundária (Figura 17) é aberta e os parâmetros geométricos, fatores de segurança próprios do método escolhido, materiais e cargas atuantes sobre a peça.

Figura 17 – Janela de inserção de parâmetros do Método da NBR 9062 para consolos curtos



A continuação a ferramenta realiza os cálculos e apresenta um relatório que aparece no campo em branco observado na Figura 18, contendo os resultados e equações utilizadas pelo método escolhido. A janela de apresentação de resultados é igual para todos os métodos.

Figura 18 – Janela de saída de resultados do Método da NBR 9062 para consolos curtos

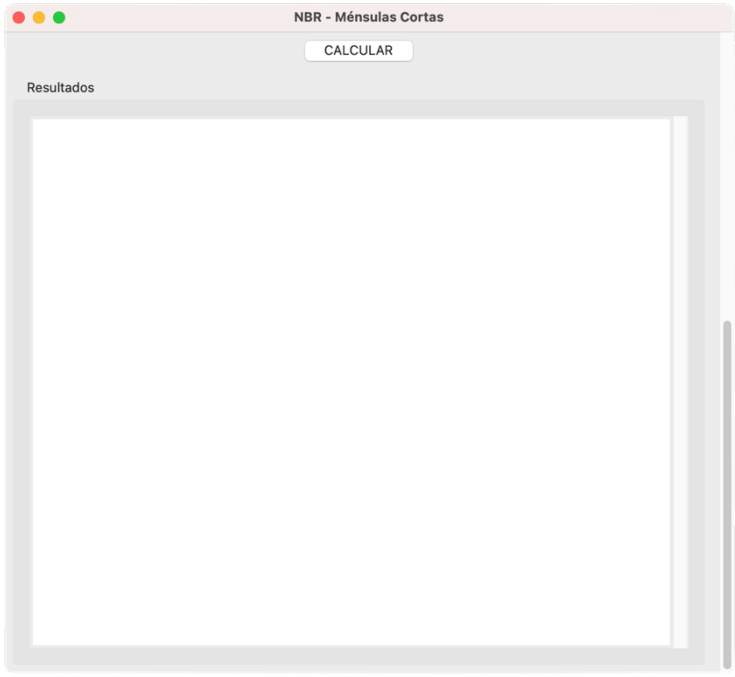


Figura 19 – Janela de inserção de parâmetros do Método da NBR 9062 para consolos muito

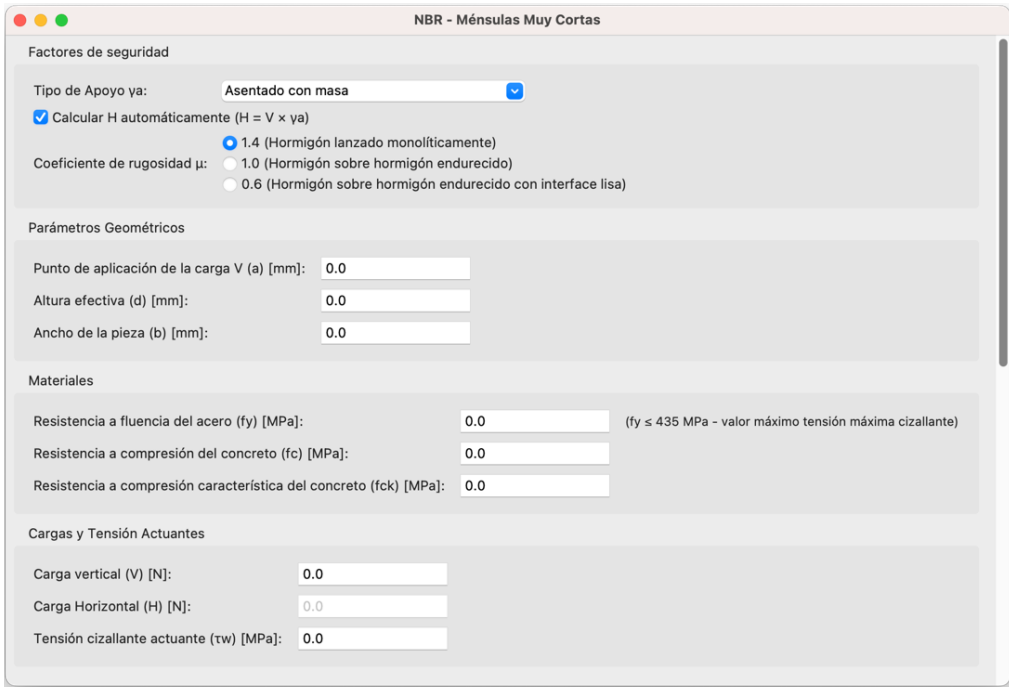


Figura 20 – Janela de inserção de parâmetros do proposto por El Debs para consolos curtos

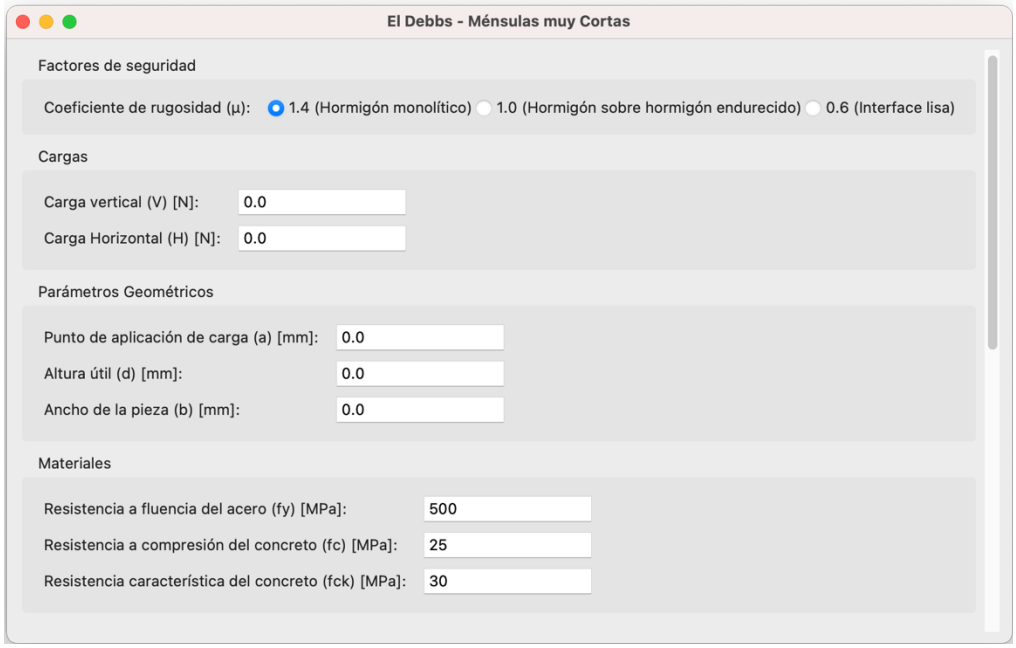
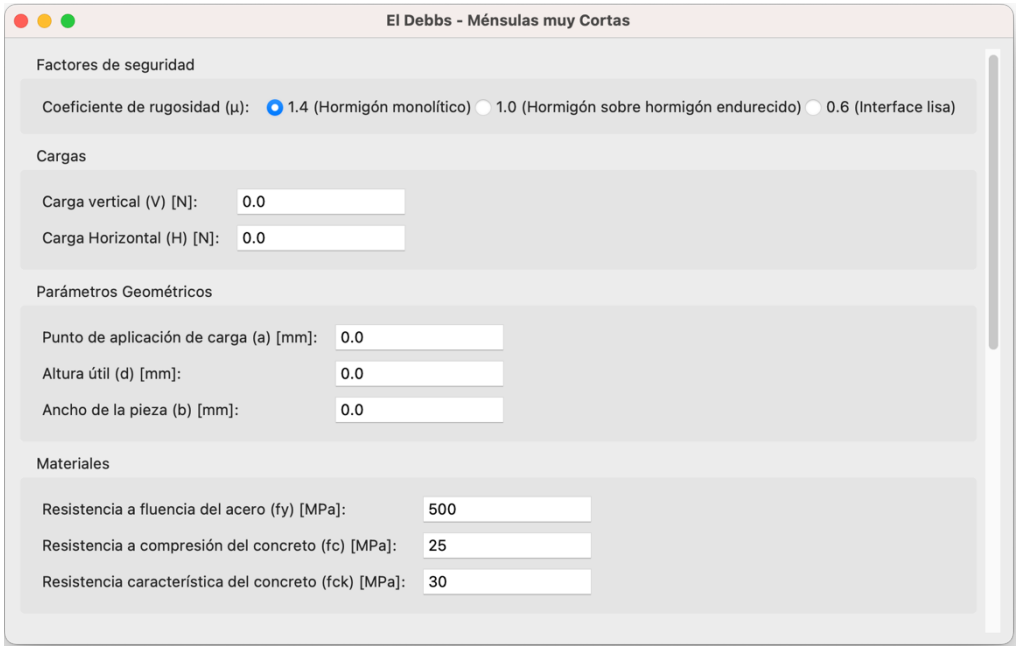


Figura 21 – Janela de inserção de parâmetros do proposto por El Debs para consolos muito curtos



5 APLICAÇÃO DA FERRAMENTA

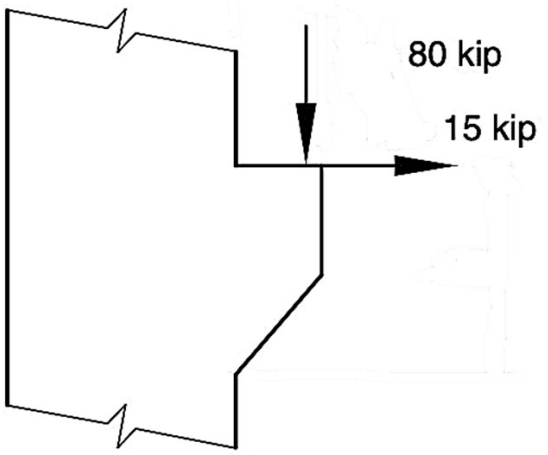
Para a aplicação da ferramenta, utilizaremos o exemplo proposto pelo PCI *Design Handbook,* na seção 5, páginas 111-115.

Um consolo submetido a uma carga vertical de 80 kip e uma horizontal de 15 kip conforme a figura.

Dimensões: $b = 14$ ", $h = 14$ ", $d = 13$ " e $l = 8$ ".

Com $fc = 5\ ksi, (concreto\ normal)$ e $fy = 60\ ksi$. Concreto lançado monoliticamente.

Figura 22 – Exemplo proposto no PCI *Design Handbook*



Fonte: Adaptado PCI *Design Handbook*, 2010

Como representação do esquema de funcionamento apresentado na Figura 15, foi utilizado o *Strut and Tie method* na ferramenta. A continuação, são apresentados os dados utilizados para cada método disponível na ferramenta.

Tabela 10 – Dimensões da peça

Dimensões	in	mm
a	6,00	152,40
l	14,00	355,60
h	14,00	355,60
d	13,00	330,20
b	14,00	355,60

Tabela 11 – Propriedades dos materiais

Materiais	ksi	MPa
fy	60,00	359,74
fc	5,00	24,63

Tabela 12 – Coeficientes de segurança

Coeficientes de segurança	(-)
γs	1,15
γc	1,40

Tabela 13 – Cargas Aplicadas

Cargas	kip	N
Vertical	80,00	355.858
Horizontal	15,00	66.723

Os valores das cargas foram assumidas como minoradas. O cálculo da tensão atuante no caso dos métodos propostos na NBR e El Debs, é realizado pela equação 8, conforme pode ser visualizado na Tabela 15. Para todos os cálculos, o valor da ação horizontal foi considerado conforme a Tabela 13.

No caso dos materiais, os coeficientes de segurança foram aplicados para os casos da NBR e dos métodos propostos por El Debs, já que nos casos dos métodos propostos pelo PCI, os coeficientes são contemplados na formulação.

Nas Figuras 25.a e 25.b é definida a geometria da treliça considerada. Uma vez resolvida a geometria, a ferramenta gera um esboço indicando as tensões atuantes nas barras da treliça conforme a Figura 23.c. A continuação, na Figura 23.d é apresentado relatório.

Figura 23.a – Inserção de parâmetros para o método e bielas e tirantes

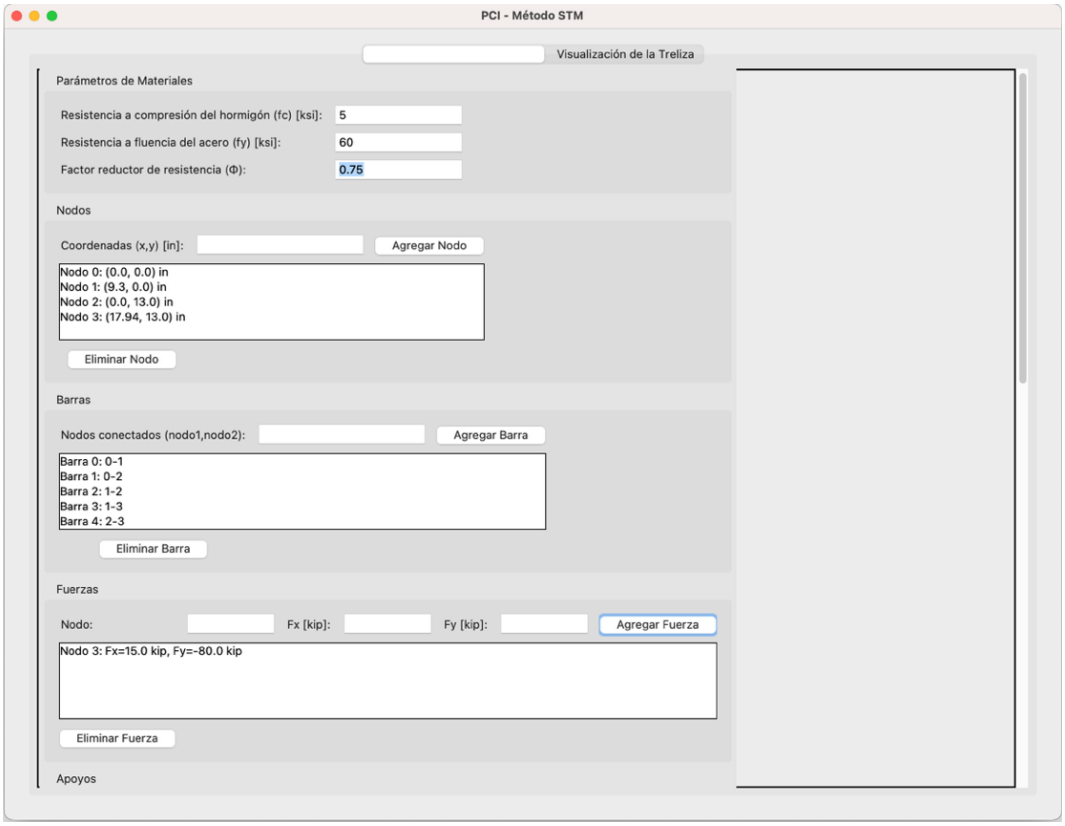


Figura 23.b – Inserção de parâmetros para o método e bielas e tirantes

Figura 23.c – Visualização da treliça

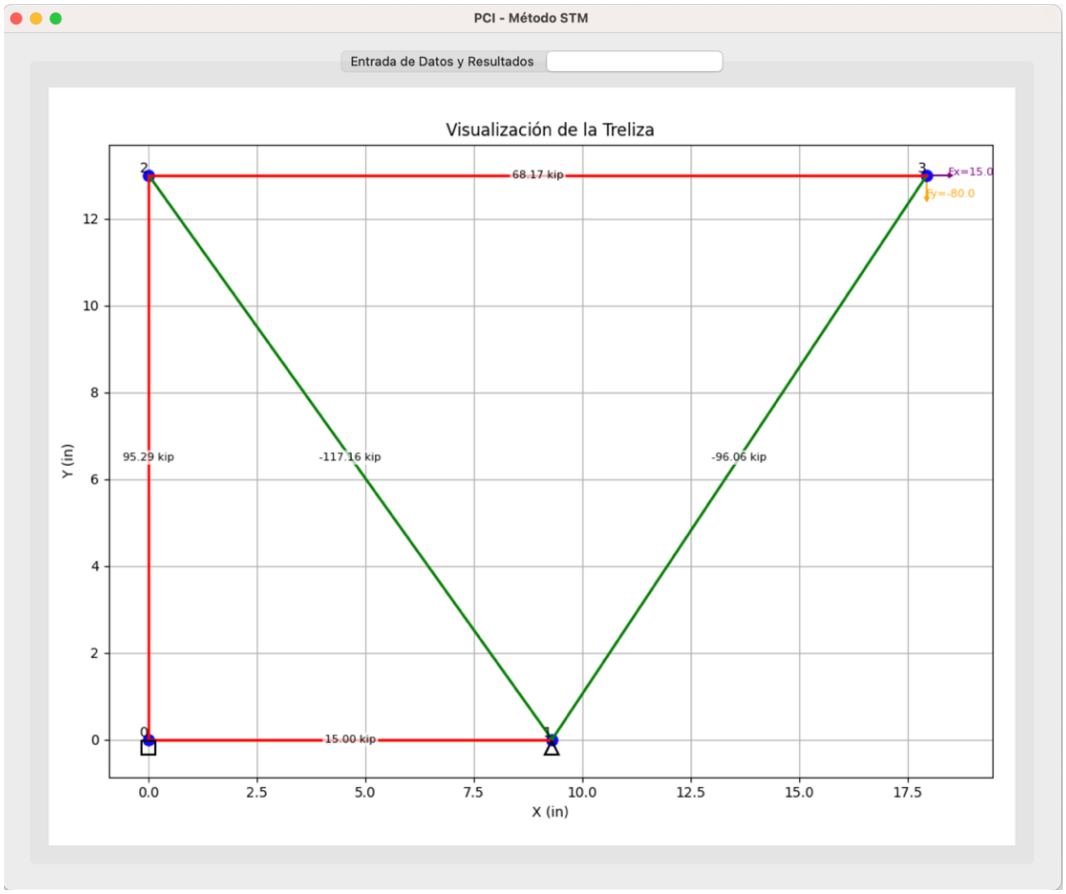
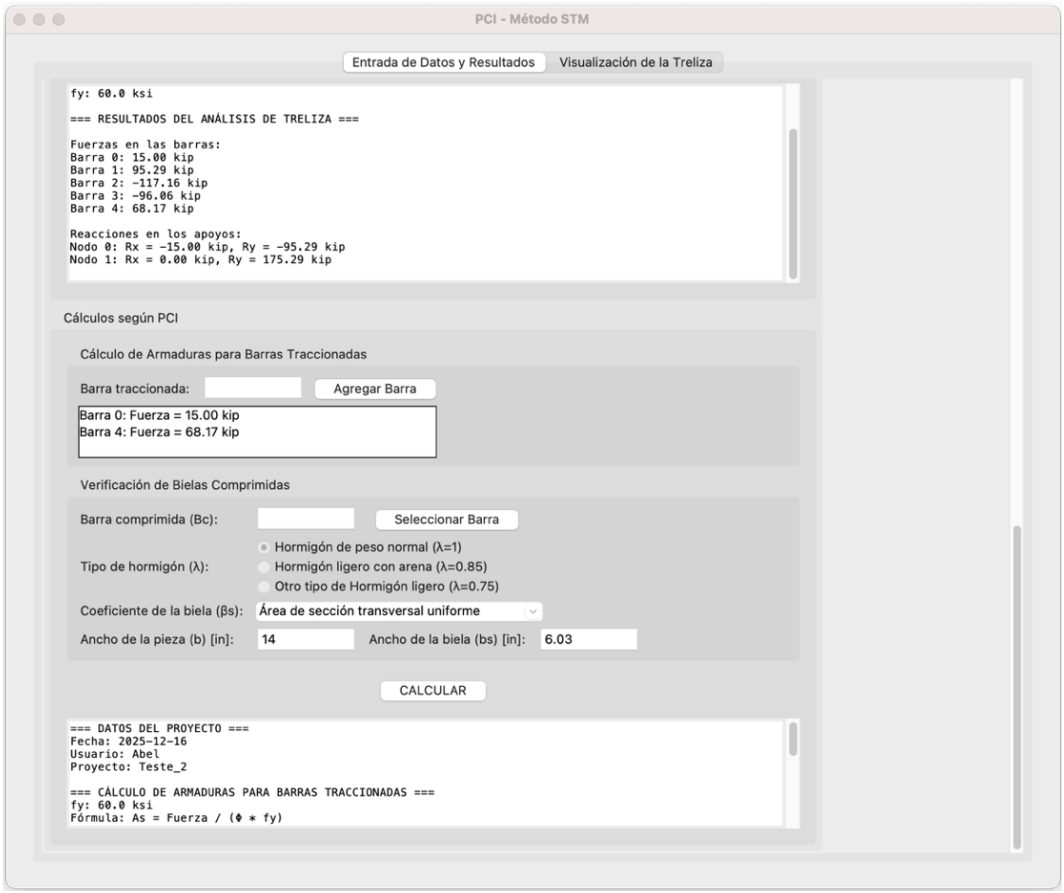


Figura 23.d – Calculo das armaduras e verificação da biela



5.1 Resultados e discussões

Tabela 14 – Área das armaduras calculadas

	PCI		NBR 9062 / NBR 6118		EI Debs	
	STM	**Viga en Balance**	**Ménsulas Cortas**	**Ménsulas Muy cortas**	**Ménsulas Cortas**	**Ménsulas Muy cortas**
Armadura Principal (mm ²)	980,64	999,99	740,95	750,73	729,86	750,73
Armadura Secundaria (mm ²)	212,90	465,80	0,67	0,85	253,63	253,64

Tabela 15 – Resistência das bielas calculadas

	PCI (*kip*)		**NBR 9062 / NBR 6118** (*MPa*)		**El Debs** (*MPa*)	
	STM	**Viga en Balance** [1]	**Ménsulas Cortas**	**Ménsulas Muy cortas**	**Ménsulas Cortas**	**Ménsulas Muy cortas**
Resistência da biela vs. Tensão atuante	161,45 > 117,16	147,00 > 80	15,25 > 3,03	3,54 > 3,03	6,22 > 3,03	3,54 > 3,03

1. Neste caso a resistência é avaliada segundo a carga máxima, conforme explicado no capítulo 3, na seção 3.2.3.2

É possível notar das equações descritas no capítulo 3, o cálculo da armadura principal para o método prescrito na NBR e o proposto pelo El Debs, é igual para o caso dos consolos muito curtos e também para a avaliação da resistência da biela de concreto. Também existe uma semelhança entre estes 2 métodos no caso dos consolos curtos.

A diferença entre a quantidade de armadura de costura deve-se ao fato de que a NBR 9062 apenas impõe o mínimo de armadura de costura por meio das equações 4 e 18. Já a ferramenta, por motivos de simplificação, assume o valor mínimo.

Nota-se também que os métodos americano proposto é mais conservador do que os modelos brasileiros.

No caso da avaliação da resistência do concreto, todos os métodos mostraram-se seguros, embora exista uma diferença marcada entre os métodos de calculo da tensão limite suportada pelo concreto.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo criar uma ferramenta computacional que realize os cálculos das áreas de armadura e verificação de resistência, e seja dependente dos parâmetros solicitados pelos modelos na Norma ABNT NBR 9062, o manual do PCI *Design Handbook* e o modelo analítico proposto por El Debs (2017).

Para isto foi revisada a bibliografia disponível sobre o assunto, de forma que seja possível a criação de um algoritmo que atenda os critérios e fazer as considerações necessárias para um funcionamento simples e interativo.

No caso do modelo de bielas e tirantes recomendado pelo PCI não foi possível criar um algoritmo que realize todos os cálculos pelo usuário pela complexidade da confeição do código para esse quesito. No entanto, a ferramenta consegue calcular uma treliça e aplicar os fatores de calculo para, uma vez que as coordenadas dos nós, as forças atuantes e os materiais considerados sejam informados, o que é de grande utilidade para a avaliação de hipóteses no momento de pesquisa.

No caso do modelo da NBR 9062, a mesma prevê os limites de resistência para a biela de concreto, o que deixa implícito que o usuário deverá realizar o calculo separadamente. Todos os demais modelos foram implementados com sucesso.

6.1 Sugestões para trabalhos futuros

A existência de uma ampla variedade de modelos de dimensionamento de consolos e os resultados obtidos da aplicação da ferramenta, mostram que os mesmos precisam ser testados e revisados. Araújo et al. (2016), consideram que os modelos normativos da ABNT e do PCI subestimam a força de ruína dos consolos e que os resultados podem ser muito divergentes entre si, em certos aspectos. Isto torna interessante a adição de novos métodos de cálculo para comparação de resultados.

Além disso, modelos com uma proposta de calculo para aplicações de carga excêntrica ou análise de torção poderiam ser adicionados numa atualização uma vez que o acontecimento deste tipo de situações na montagem da estrutura é muito provável e indesejável.

Outro ponto passível de ser abordado são apoios, os quais são de vital importância para a transmissão de cargas. Alguns modelos dependem do mesmo para

a determinação da largura da biela, além de variar os coeficientes conforme a capacidade de transmissão de carga do apoio utilizado.

Novas tecnologias que visam um melhor comportamento estrutural, economia e praticidade poderiam mudar os parâmetros e os critérios de dimensionamento. Costa (2009), mostra que a adição de fibras metálicas em consolos montados em duas etapas, pode melhorar a ductilidade da peça sem perder trabalhabilidade.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, Daniel et al. Análise Comparativa de modelos de calculo para consolos de concreto. Revista Ibracon De Estruturas e Materiais, [s. l.], v. 9, n. 3, p. 435-470, 3 jun. 2016. DOI https://doi.org/10.1590/S1983-41952016000300007. Disponível em: https://www.scielo.br/j/riem/a/n5zHkDwRxXJKyx4TWFk4ccs/?lang=pt. Acesso em: 5 nov. 2025.

ARAÚJO, Daniel; AZEVEDO, Sergio; OLIVEIRA, Edilene; JUNIOR, Luiz. Avaliação da resistência de consolos de concreto moldados em etapa distinta do pilar. Revista Ibracon De Estruturas e Materiais, [s. l.], v. 10, n. 2, p. 509-546, 17 abr. 2017. DOI http://dx.doi.org/10.1590/S1983-41952017000200011. Disponível em: https://revistas.ibracon.org.br/index.php/riem/article/view/822. Acesso em: 5 nov. 2025.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS NBR 6118: Projeto de estruturas de concreto - Procedimento. Terceira. ed. [*S. l.*: *s. n.*], 2014.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS NBR 9062: Projeto e execução de estruturas de concreto pré-moldado. Terceira. ed. [*S. l.*: *s. n.*], 2017.

BRITISH STANDARD INSTITUTION (Reino Unido). BS 8110-1: Structural use of concrete — Part 1: Code of practice for design and construction. Primeira. ed. [*S. l.*: *s. n.*], 1997.

CANHA, Rejane Martins Fernandes et al. Numerical analysis of reinforced high strength concrete corbels. Engineering Structures, [s. l.], v. 74, p. 130-144, 6 jun. 2014. DOI https://doi.org/10.1016/j.engstruct.2014.05.014. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0141029614003010. Acesso em: 12 nov. 2025.

CARVALHO, D; TAPAJÓS, L; MAUÉS, F; FERREIRA, M. ANÁLISE COMPUTACIONAL DE CONSOLOS DE CONCRETO ARMADO DE ALTA RESISTÊNCIA. Ibero-Latin American Congress on Computational Methods in Engineering (CILAMCE 2016), [*s. l.*], 2016.

COSTA, Jônatas Barreto de Andrade. Estudo experimental de consolos de concreto com fibras moldadas em etapas distintas dos pilares. 2009. Dissertação (Mestrado) – Universidade de São Paulo, São Carlos, 2009. Disponível em: http://www.teses.usp.br/teses/disponiveis/18/18134/tde-14072009-170612/. Acesso em: 6 nov. 2025.

EL DEBS, Mounir. Concreto Pré-moldado: Fundamentos e Aplicações. Segunda. ed. [*S. l.*: *s. n.*], 2017.

ELLIOTT, Kim. Precast Concrete Structures. Primeira. ed. [*S. l.*]: Butterworth-Heinemann, 2002.

LEONHARDT, Fritz; MÖNNIG, Eduard. Construções de Concreto: Casos especiais de dimensionamento de estruturas de concreto armado. Primeira. ed. [*S. l.*: *s. n.*]: Interciência Ltda., 1978. v. 2.

LEONHARDT, Fritz; MÖNNIG, Eduard. Construções de Concreto: Princípios Básicos sobre a armação de estruturas de concreto armado. Primeira. ed. [*S. l.*]: Interciência Ltda., 1978. v. 3.

MACHADO, Claudinei; PIMENTA, Paulo. Consolos Muito Curtos de Concreto Armado: Modelos e Critérios para a Análise com uma Nova Formulação Proposta de Atrito-Cisalhamento. IV SIMPÓSIO EPUSP SOBRE ESTRUTURAS DE CONCRETO, [s. l.], 2000.

MATTOCK, Alan; CHEN, K; SOONGSWANG, K. The Behavior of Reinforced Concrete Corbels. PCI Journal, [s. l.], v. 9, n. 2, p. 52-77, 1976. DOI https://doi.org/10.15554/pcij.03011976.52.77. Disponível em: https://www.pci.org/PCI/PCI/Publications/PCI_Journal/Issues/1976/March-April/The_Behavior_of_Reinforced_Concrete_Corbels.aspx. Acesso em: 5 nov. 2025.

OLIVEIRA, Edilene. CONSOLOS DE CONCRETO MOLDADOS EM DUAS ETAPAS: Influência do tratamento da interface e da adição de fibras de aço. 2012. Dissertação (Mestre em Engenharia Civil) - Universidade Federal de Goiás, Programa de Pós-Graduação em Geotecnia, Estruturas e Construção Civil, [S. l.], 2012.

PRECAST/PRESTRESSED CONCRETE INSTITUTE (USA). PCI DESIGN HANDBOOK. *In*: PRECAST AND PRESTRESSED CONCRETE. Sétima . ed. [*S. l.*: *s. n.*], 2010.

PRADO, Lisiane et al. Avaliação do comportamento de ligação de montagem viga-pilar para estruturas de concreto pré-moldado. Associação Nacional de Tecnologia do Ambiente Construído, [s. l.], 24 out. 2017.

REGINATO, Luan. CONTRIBUIÇÃO AO PROJETO DE CONSOLOS DE CONCRETO COM BASE EM SIMULAÇÕES NUMÉRICAS. 2020. Dissertação (Mestre em Ciências) - UNIVERSIDADE DE SÃO PAULO, ESCOLA DE ENGENHARIA DE SÃO CARLOS, [S. l.], 2020.

RUSSO, Gaetano; VENIR, Raffaele; PAULETTA, Margherita; SOMMA, Giuliana. Reinforced Concrete Corbels: Shear Strength Model and Design Formula. ACI STRUCTURAL JOURNAL, [s. l.], v. 103, p. 3-10, 1 jan. 2006.

SOLANKI, Himat; SABNIS, Garajanan. Reinforced Concrete Corbels: Simplified. ACI STRUCTURAL JOURNAL, [s. l.], v. 84, ed. 5, p. 428-432, 1 set. 1987. DOI 10.14359/15164. Disponível em: https://www.concrete.org/publications/internationalconcreteabstractsportal.aspx?m=details&ID=15164. Acesso em: 11 nov. 2025.

TEODORO, Luiz Fernando. Estudo numérico do comportamento semirrígido de uma ligação viga-pilar pré-moldada com concreto com fibras no consolo e no dente da viga. 2022. Dissertação (Mestre em Ciências) – Universidade de São Paulo, Escola de Engenharia de São Carlos, [S. l.], 2022.

TORRES, Fernando. Análise Teórico Experimental de Consolos de Concreto Armado. 1998. Dissertação (Mestrado em Engenharia de Estruturas) - Escola de Engenharia de São Carlos da Universidade de São Paulo, [*S. l.*], 2017.

WISSMAN, Jorge; SILVEIRA, J; RHUNKE, L; LEITE, J; VIZINI, D. Análise de desabamento com vítimas fatais, durante concretagem, devido a ruptura de consolo pré-moldado. Revista Brasileira de Criminalística, [s. l.], v. 12, n. 4, p. 24-30, 11 ago. 2023.

DOI http://dx.doi.org/10.15260/rbc.v12i4.684. Disponível em: https://revista.rbc.org.br/index.php/rbc/article/view/684. Acesso em: 11 nov. 2025.

8 APÊNDICE 1: CÓDIGO COMPUTACIONAL

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use("TkAgg")  # Use TkAgg backend for GUI
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import tkinter as tk
from tkinter import ttk, messagebox, scrolledtext
from PIL import Image, ImageTk
import threading
from datetime import date
import math

def solve_truss(nodes, bars, forces, supports):
    num_nodes = len(nodes)
    num_bars = len(bars)

    # Grados de libertad por nodo (2 por nodo: desplazamiento en X,
desplazamiento en Y)
    dof_per_node = 2
    total_dof = num_nodes * dof_per_node

    # Inicializar la matriz de rigidez global (K) y el vector de fuerzas
(F)
    K = np.zeros((total_dof, total_dof))
    F = np.zeros(total_dof)

    # Ensamblar el vector de fuerzas F
    for node_idx, (fx, fy) in forces.items():
        F[node_idx * dof_per_node] += fx
        F[node_idx * dof_per_node + 1] += fy

    # Ensamblar la matriz de rigidez global K
    bar_lengths = {}
    bar_angles = {}

    for i, (node1_idx, node2_idx) in enumerate(bars):
        x1, y1 = nodes[node1_idx]
        x2, y2 = nodes[node2_idx]

        dx = x2 - x1
        dy = y2 - y1

        length = np.sqrt(dx**2 + dy**2)
        angle = np.arctan2(dy, dx)

        bar_lengths[i] = length
        bar_angles[i] = angle

        cos_a = np.cos(angle)
        sin_a = np.sin(angle)

        AE_L = 1.0 / length
```

```
            k_bar = AE_L * np.array([
                [cos_a**2, cos_a*sin_a, -cos_a**2, -cos_a*sin_a],
                [cos_a*sin_a, sin_a**2, -cos_a*sin_a, -sin_a**2],
                [-cos_a**2, -cos_a*sin_a, cos_a**2, cos_a*sin_a],
                [-cos_a*sin_a, -sin_a**2, cos_a*sin_a, sin_a**2]
            ])

            dofs = [
                node1_idx * dof_per_node,
                node1_idx * dof_per_node + 1,
                node2_idx * dof_per_node,
                node2_idx * dof_per_node + 1
            ]

            for r in range(4):
                for c in range(4):
                    K[dofs[r], dofs[c]] += k_bar[r, c]

    # Aplicar condiciones de contorno (apoyos)
    restrained_dofs = []
    for node_idx, support_type in supports.items():
        if support_type == 'fijo':
            restrained_dofs.append(node_idx * dof_per_node)
            restrained_dofs.append(node_idx * dof_per_node + 1)
        elif support_type == 'movil_x':
            restrained_dofs.append(node_idx * dof_per_node + 1)
        elif support_type == 'movil_y':
            restrained_dofs.append(node_idx * dof_per_node)
        elif support_type == 'empotrado':
            restrained_dofs.append(node_idx * dof_per_node)
            restrained_dofs.append(node_idx * dof_per_node + 1)

    free_dofs = [dof for dof in range(total_dof) if dof not in
restrained_dofs]

    K_red = K[np.ix_(free_dofs, free_dofs)]
    F_red = F[free_dofs]

    U_red = np.linalg.solve(K_red, F_red)

    U = np.zeros(total_dof)
    for i, dof in enumerate(free_dofs):
        U[dof] = U_red[i]

    # Calcular fuerzas en las barras
    bar_forces = {}
    for i, (node1_idx, node2_idx) in enumerate(bars):
        length = bar_lengths[i]
        angle = bar_angles[i]

        cos_a = np.cos(angle)
        sin_a = np.sin(angle)

        u1x = U[node1_idx * dof_per_node]
        u1y = U[node1_idx * dof_per_node + 1]
        u2x = U[node2_idx * dof_per_node]
        u2y = U[node2_idx * dof_per_node + 1]
```

```
            delta_L = (u2x - u1x) * cos_a + (u2y - u1y) * sin_a
            force = (1.0 / length) * delta_L
            bar_forces[i] = force

        # Calcular reacciones en los apoyos
        reactions = {}
        for node_idx, support_type in supports.items():
            Rx = 0.0
            Ry = 0.0

            dof_x = node_idx * dof_per_node
            dof_y = node_idx * dof_per_node + 1

            if support_type in ['fijo', 'movil_y', 'empotrado']:
                Rx = np.dot(K[dof_x, :], U) - F[dof_x]

            if support_type in ['fijo', 'movil_x', 'empotrado']:
                Ry = np.dot(K[dof_y, :], U) - F[dof_y]

            reactions[node_idx] = (Rx, Ry)

        return bar_forces, reactions

class TrussCalculatorGUI:
    def __init__(self, root, project_info=None):
        self.root = root
        self.root.title("PCI - Método STM")
        self.root.geometry("1200x900")
        self.project_info = project_info if project_info else {}

        # Variables para almacenar datos
        self.nodes = []
        self.bars = []
        self.forces = {}
        self.supports = {}
        self.bar_forces = {}
        self.reactions = {}

        # Variables para parámetros adicionales
        self.fc = tk.DoubleVar(value=3.625)  # 25 MPa = 3.625 ksi
        self.fy = tk.DoubleVar(value=72.5)   # 500 MPa = 72.5 ksi
        self.fR = tk.DoubleVar(value=0.75)   # Generalmente 0.75
        self.selected_tension_bars = []  # Lista para barras traccionadas
seleccionadas
        self.selected_compression_bar = None  # Barra comprimida
seleccionada

        self.create_widgets()

    def create_widgets(self):
        # Crear notebook para pestañas
        notebook = ttk.Notebook(self.root)
        notebook.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Pestaña de entrada de datos y resultados
        self.data_results_frame = ttk.Frame(notebook)
```

```
        notebook.add(self.data_results_frame, text="Entrada de Datos y
Resultados")

        # Pestaña de visualización
        self.plot_frame = ttk.Frame(notebook)
        notebook.add(self.plot_frame, text="Visualización de la Treliza")

        self.create_data_results_widgets()
        self.create_plot_widgets()

    def create_data_results_widgets(self):
        # Frame principal con scroll
        canvas = tk.Canvas(self.data_results_frame)
        scrollbar = ttk.Scrollbar(self.data_results_frame,
orient="vertical", command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )
        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        # Parámetros de materiales
        materials_frame = ttk.LabelFrame(scrollable_frame, text="Parámetros
de Materiales", padding=10)
        materials_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(materials_frame, text="Resistencia a compresión del
hormigón (fc) [ksi]:").grid(row=0, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(materials_frame, textvariable=self.fc,
width=15).grid(row=0, column=1, padx=5, pady=2)

        ttk.Label(materials_frame, text="Resistencia a fluencia del acero
(fy) [ksi]:").grid(row=1, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(materials_frame, textvariable=self.fy,
width=15).grid(row=1, column=1, padx=5, pady=2)

        ttk.Label(materials_frame, text="Factor reductor de resistencia
(Φ):").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(materials_frame, textvariable=self.fR,
width=15).grid(row=2, column=1, padx=5, pady=2)

        # Nodos
        nodes_frame = ttk.LabelFrame(scrollable_frame, text="Nodos",
padding=10)
        nodes_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(nodes_frame, text="Coordenadas (x,y) [in]:").grid(row=0,
column=0, sticky=tk.W, padx=5, pady=2)
        self.node_entry = ttk.Entry(nodes_frame, width=20)
        self.node_entry.grid(row=0, column=1, padx=5, pady=2)
        ttk.Button(nodes_frame, text="Agregar Nodo",
command=self.add_node).grid(row=0, column=2, padx=5, pady=2)

        self.nodes_listbox = tk.Listbox(nodes_frame, height=5)
```

```
        self.nodes_listbox.grid(row=1, column=0, columnspan=3,
sticky=tk.EW, padx=5, pady=5)
        ttk.Button(nodes_frame, text="Eliminar Nodo",
command=self.remove_node).grid(row=2, column=0, padx=5, pady=2)

        # Barras
        bars_frame = ttk.LabelFrame(scrollable_frame, text="Barras",
padding=10)
        bars_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(bars_frame, text="Nodos conectados
(nodo1,nodo2):").grid(row=0, column=0, sticky=tk.W, padx=5, pady=2)
        self.bar_entry = ttk.Entry(bars_frame, width=20)
        self.bar_entry.grid(row=0, column=1, padx=5, pady=2)
        ttk.Button(bars_frame, text="Agregar Barra",
command=self.add_bar).grid(row=0, column=2, padx=5, pady=2)

        self.bars_listbox = tk.Listbox(bars_frame, height=5)
        self.bars_listbox.grid(row=1, column=0, columnspan=3, sticky=tk.EW,
padx=5, pady=5)
        ttk.Button(bars_frame, text="Eliminar Barra",
command=self.remove_bar).grid(row=2, column=0, padx=5, pady=2)

        # Fuerzas
        forces_frame = ttk.LabelFrame(scrollable_frame, text="Fuerzas",
padding=10)
        forces_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(forces_frame, text="Nodo:").grid(row=0, column=0,
sticky=tk.W, padx=5, pady=2)
        self.force_node_entry = ttk.Entry(forces_frame, width=10)
        self.force_node_entry.grid(row=0, column=1, padx=5, pady=2)

        ttk.Label(forces_frame, text="Fx [kip]:").grid(row=0, column=2,
sticky=tk.W, padx=5, pady=2)
        self.fx_entry = ttk.Entry(forces_frame, width=10)
        self.fx_entry.grid(row=0, column=3, padx=5, pady=2)

        ttk.Label(forces_frame, text="Fy [kip]:").grid(row=0, column=4,
sticky=tk.W, padx=5, pady=2)
        self.fy_entry = ttk.Entry(forces_frame, width=10)
        self.fy_entry.grid(row=0, column=5, padx=5, pady=2)

        ttk.Button(forces_frame, text="Agregar Fuerza",
command=self.add_force).grid(row=0, column=6, padx=5, pady=2)

        self.forces_listbox = tk.Listbox(forces_frame, height=5)
        self.forces_listbox.grid(row=1, column=0, columnspan=7,
sticky=tk.EW, padx=5, pady=5)
        ttk.Button(forces_frame, text="Eliminar Fuerza",
command=self.remove_force).grid(row=2, column=0, padx=5, pady=2)

        # Apoyos
        supports_frame = ttk.LabelFrame(scrollable_frame, text="Apoyos",
padding=10)
        supports_frame.pack(fill=tk.X, padx=5, pady=5)
```

```
        ttk.Label(supports_frame, text="Nodo:").grid(row=0, column=0,
sticky=tk.W, padx=5, pady=2)
        self.support_node_entry = ttk.Entry(supports_frame, width=10)
        self.support_node_entry.grid(row=0, column=1, padx=5, pady=2)

        ttk.Label(supports_frame, text="Tipo:").grid(row=0, column=2,
sticky=tk.W, padx=5, pady=2)
        self.support_type_var = tk.StringVar(value="fijo")
        support_combo = ttk.Combobox(supports_frame,
textvariable=self.support_type_var,
                                     values=["fijo", "movil_x", "movil_y",
"empotrado"], width=15)
        support_combo.grid(row=0, column=3, padx=5, pady=2)

        ttk.Button(supports_frame, text="Agregar Apoyo",
command=self.add_support).grid(row=0, column=4, padx=5, pady=2)

        self.supports_listbox = tk.Listbox(supports_frame, height=5)
        self.supports_listbox.grid(row=1, column=0, columnspan=5,
sticky=tk.EW, padx=5, pady=5)
        ttk.Button(supports_frame, text="Eliminar Apoyo",
command=self.remove_support).grid(row=2, column=0, padx=5, pady=2)

        # Botón de cálculo
        calc_frame = ttk.Frame(scrollable_frame)
        calc_frame.pack(fill=tk.X, padx=5, pady=20)

        ttk.Button(calc_frame, text="CALCULAR TRELIZA",
command=self.calculate_truss,
                   style="Accent.TButton").pack(pady=10)

        # Resultados principales
        main_results_frame = ttk.LabelFrame(scrollable_frame,
text="Resultados Principales", padding=10)
        main_results_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.results_text = scrolledtext.ScrolledText(main_results_frame,
height=15, width=80)
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Cálculos según PCI
        additional_frame = ttk.LabelFrame(scrollable_frame, text="Cálculos
según PCI", padding=10)
        additional_frame.pack(fill=tk.X, padx=5, pady=5)

        # Cálculo de armaduras para barras traccionadas
        armadura_frame = ttk.LabelFrame(additional_frame, text="Cálculo de
Armaduras para Barras Traccionadas", padding=5)
        armadura_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(armadura_frame, text="Barra traccionada:").grid(row=0,
column=0, sticky=tk.W, padx=5, pady=2)
        self.barra_traccionada_entry = ttk.Entry(armadura_frame, width=10)
        self.barra_traccionada_entry.grid(row=0, column=1, padx=5, pady=2)

        ttk.Button(armadura_frame, text="Agregar Barra",
command=self.add_tension_bar).grid(row=0, column=2, padx=5, pady=2)
```

```
        self.tension_bars_listbox = tk.Listbox(armadura_frame, height=3)
        self.tension_bars_listbox.grid(row=1, column=0, columnspan=4,
sticky=tk.EW, padx=5, pady=2)

        # Verificación de bielas
        biela_frame = ttk.LabelFrame(additional_frame, text="Verificación
de Bielas Comprimidas", padding=5)
        biela_frame.pack(fill=tk.X, padx=5, pady=5)

        # Selección de barra comprimida
        ttk.Label(biela_frame, text="Barra comprimida (Bc):").grid(row=0,
column=0, sticky=tk.W, padx=5, pady=2)
        self.barra_comprimida_entry = ttk.Entry(biela_frame, width=10)
        self.barra_comprimida_entry.grid(row=0, column=1, padx=5, pady=2)
        ttk.Button(biela_frame, text="Seleccionar Barra",
command=self.select_compression_bar).grid(row=0, column=2, padx=5, pady=2)

        # Tipo de hormigón (λ)
        ttk.Label(biela_frame, text="Tipo de hormigón (λ):").grid(row=1,
column=0, sticky=tk.W, padx=5, pady=2)
        self.concrete_type_var = tk.StringVar(value="Hormigón de peso
normal")
        concrete_type_frame = ttk.Frame(biela_frame)
        concrete_type_frame.grid(row=1, column=1, columnspan=3,
sticky=tk.W, padx=5, pady=2)
        ttk.Radiobutton(concrete_type_frame, text="Hormigón de peso normal
(λ=1)",
                        variable=self.concrete_type_var, value="Hormigón de
peso normal").pack(anchor=tk.W)
        ttk.Radiobutton(concrete_type_frame, text="Hormigón ligero con
arena (λ=0.85)",
                        variable=self.concrete_type_var, value="Hormigón
ligero con arena").pack(anchor=tk.W)
        ttk.Radiobutton(concrete_type_frame, text="Otro tipo de Hormigón
ligero (λ=0.75)",
                        variable=self.concrete_type_var, value="Otro tipo de
Hormigón ligero").pack(anchor=tk.W)

        # Coeficiente de la biela (βs)
        ttk.Label(biela_frame, text="Coeficiente de la biela
(βs):").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        self.beta_s_var = tk.StringVar(value="Área de sección transversal
uniforme")
        beta_s_combo = ttk.Combobox(biela_frame,
textvariable=self.beta_s_var,
                                    values=["Área de sección transversal
uniforme",
                                            "Formato de botella con
armadura",
                                            "Formato de botella sin
armadura",
                                            "Bielas en zonas de tensión",
                                            "Otros"], width=30)
        beta_s_combo.grid(row=2, column=1, columnspan=3, sticky=tk.W,
padx=5, pady=2)
```

```
        # Parámetros geométricos
        ttk.Label(biela_frame, text="Ancho de la pieza (b)
[in]:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=2)
        self.b_var = tk.DoubleVar(value=11.81)  # 0.3 m = 11.81 in
        ttk.Entry(biela_frame, textvariable=self.b_var,
width=10).grid(row=3, column=1, padx=5, pady=2)

        ttk.Label(biela_frame, text="Ancho de la biela (bs)
[in]:").grid(row=3, column=2, sticky=tk.W, padx=5, pady=2)
        self.bs_var = tk.DoubleVar(value=7.87)  # 0.2 m = 7.87 in
        ttk.Entry(biela_frame, textvariable=self.bs_var,
width=10).grid(row=3, column=3, padx=5, pady=2)

        # Botón CALCULAR unificado
        calc_button_frame = ttk.Frame(additional_frame)
        calc_button_frame.pack(fill=tk.X, padx=5, pady=5)
        ttk.Button(calc_button_frame, text="CALCULAR",
command=self.calculate_pci, style="Accent.TButton").pack(pady=5)

        self.additional_results_text =
scrolledtext.ScrolledText(additional_frame, height=8, width=80)
        self.additional_results_text.pack(fill=tk.X, padx=5, pady=5)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

    def create_plot_widgets(self):
        # Frame para el gráfico
        self.fig = Figure(figsize=(10, 8), dpi=100)
        self.ax = self.fig.add_subplot(111)

        self.canvas = FigureCanvasTkAgg(self.fig, self.plot_frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True,
padx=10, pady=10)

    def add_node(self):
        try:
            coords = self.node_entry.get().strip()
            if not coords:
                return
            x, y = map(float, coords.split(","))
            self.nodes.append((x, y))
            self.nodes_listbox.insert(tk.END, f"Nodo {len(self.nodes)-1}:
({x}, {y}) in")
            self.node_entry.delete(0, tk.END)
        except ValueError:
            messagebox.showerror("Error", "Formato inválido. Use: x,y")

    def remove_node(self):
        selection = self.nodes_listbox.curselection()
        if selection:
            index = selection[0]
            self.nodes.pop(index)
            self.nodes_listbox.delete(0, tk.END)
            for i, (x, y) in enumerate(self.nodes):
                self.nodes_listbox.insert(tk.END, f"Nodo {i}: ({x}, {y})
in")
```

```
    def add_bar(self):
        try:
            bar_data = self.bar_entry.get().strip()
            if not bar_data:
                return
            n1, n2 = map(int, bar_data.split(","))
            if 0 <= n1 < len(self.nodes) and 0 <= n2 < len(self.nodes):
                self.bars.append((n1, n2))
                self.bars_listbox.insert(tk.END, f"Barra {len(self.bars)-
1}: {n1}-{n2}")
                self.bar_entry.delete(0, tk.END)
            else:
                messagebox.showerror("Error", "Índices de nodo inválidos.")
        except ValueError:
            messagebox.showerror("Error", "Formato inválido. Use:
nodo1,nodo2")

    def remove_bar(self):
        selection = self.bars_listbox.curselection()
        if selection:
            index = selection[0]
            self.bars.pop(index)
            self.bars_listbox.delete(0, tk.END)
            for i, (n1, n2) in enumerate(self.bars):
                self.bars_listbox.insert(tk.END, f"Barra {i}: {n1}-{n2}")

    def add_force(self):
        try:
            node_idx = int(self.force_node_entry.get())
            fx = float(self.fx_entry.get())
            fy = float(self.fy_entry.get())

            if 0 <= node_idx < len(self.nodes):
                self.forces[node_idx] = (fx, fy)
                self.update_forces_listbox()
                self.force_node_entry.delete(0, tk.END)
                self.fy_entry.delete(0, tk.END)
                self.fx_entry.delete(0, tk.END)
            else:
                messagebox.showerror("Error", "Índice de nodo inválido.")
        except ValueError:
            messagebox.showerror("Error", "Formato inválido. Asegúrese de
que los valores sean numéricos.")

    def remove_force(self):
        selection = self.forces_listbox.curselection()
        if selection:
            node_idx_to_remove =
int(self.forces_listbox.get(selection[0]).split(":")[0].split(" ")[1])
            if node_idx_to_remove in self.forces:
                del self.forces[node_idx_to_remove]
                self.update_forces_listbox()

    def update_forces_listbox(self):
        self.forces_listbox.delete(0, tk.END)
        for node_idx, (fx, fy) in self.forces.items():
```

```
            self.forces_listbox.insert(tk.END, f"Nodo {node_idx}: Fx={fx}
kip, Fy={fy} kip")

    def add_support(self):
        try:
            node_idx = int(self.support_node_entry.get())
            support_type = self.support_type_var.get()

            if 0 <= node_idx < len(self.nodes):
                self.supports[node_idx] = support_type
                self.update_supports_listbox()
                self.support_node_entry.delete(0, tk.END)
            else:
                messagebox.showerror("Error", "Índice de nodo inválido.")
        except ValueError:
            messagebox.showerror("Error", "Formato inválido. Asegúrese de
que el nodo sea numérico.")

    def remove_support(self):
        selection = self.supports_listbox.curselection()
        if selection:
            node_idx_to_remove =
int(self.supports_listbox.get(selection[0]).split(":")[0].split(" ")[1])
            if node_idx_to_remove in self.supports:
                del self.supports[node_idx_to_remove]
                self.update_supports_listbox()

    def update_supports_listbox(self):
        self.supports_listbox.delete(0, tk.END)
        for node_idx, support_type in self.supports.items():
            self.supports_listbox.insert(tk.END, f"Nodo {node_idx}:
{support_type}")

    def add_tension_bar(self):
        try:
            bar_idx = int(self.barra_traccionada_entry.get())
            if bar_idx not in self.bar_forces:
                messagebox.showerror("Error", "Índice de barra inválido.")
                return

            if bar_idx in self.selected_tension_bars:
                messagebox.showwarning("Advertencia", f"La barra {bar_idx}
ya está en la lista.")
                return

            self.selected_tension_bars.append(bar_idx)
            self.update_tension_bars_listbox()
            self.barra_traccionada_entry.delete(0, tk.END)

        except ValueError:
            messagebox.showerror("Error", "Por favor ingrese un índice de
barra válido.")

    def update_tension_bars_listbox(self):
        self.tension_bars_listbox.delete(0, tk.END)
        for bar_idx in self.selected_tension_bars:
            force = self.bar_forces.get(bar_idx, 0)
```

```
            self.tension_bars_listbox.insert(tk.END, f"Barra {bar_idx}:
Fuerza = {force:.2f} kip")

    def select_compression_bar(self):
        try:
            bar_idx = int(self.barra_comprimida_entry.get())
            if bar_idx not in self.bar_forces:
                messagebox.showerror("Error", "Índice de barra inválido.")
                return

            force = self.bar_forces[bar_idx]
            if force >= 0:
                messagebox.showwarning("Advertencia", f"La barra {bar_idx}
no está comprimida (fuerza = {force:.2f} kip)")
                return

            self.selected_compression_bar = bar_idx
            self.barra_comprimida_entry.delete(0, tk.END)
            messagebox.showinfo("Info", f"Barra {bar_idx} seleccionada como
biela comprimida (Bc = {force:.2f} kip)")

        except ValueError:
            messagebox.showerror("Error", "Por favor ingrese un índice de
barra válido.")

    def calculate_truss(self):
        try:
            if not self.nodes or not self.bars:
                messagebox.showwarning("Advertencia", "Debe ingresar nodos
y barras para calcular la treliza.")
                return

            self.bar_forces, self.reactions = solve_truss(self.nodes,
self.bars, self.forces, self.supports)

            # Mostrar datos iniciales primero
            results = f"=== DATOS INICIALES ===\n"
            results += f"Nombre del usuario:
{self.project_info.get('user_name', 'N/A')}\n"
            results += f"Nombre del proyecto:
{self.project_info.get('project_name', 'N/A')}\n"
            results += f"Fecha: {self.project_info.get('date', 'N/A')}\n"
            results += f"fc: {self.fc.get()} ksi\n"
            results += f"fy: {self.fy.get()} ksi\n\n"

            results += "=== RESULTADOS DEL ANÁLISIS DE TRELIZA ===\n\n"
            results += "Fuerzas en las barras:\n"
            for bar_idx, force in self.bar_forces.items():
                results += f"Barra {bar_idx}: {force:.2f} kip\n"

            results += "\nReacciones en los apoyos:\n"
            for node_idx, (rx, ry) in self.reactions.items():
                results += f"Nodo {node_idx}: Rx = {rx:.2f} kip, Ry =
{ry:.2f} kip\n"

            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, results)
```

```
            self.plot_truss()

        except Exception as e:
            messagebox.showerror("Error de Cálculo", f"Ocurrió un error
durante el cálculo: {e}")

    def calculate_pci(self):
        """Función unificada para calcular tanto armaduras como
verificación de bielas"""
        self.calcular_armadura()
        self.verify_struts()

    def calcular_armadura(self):
        try:
            if not self.bar_forces:
                messagebox.showwarning("Advertencia", "Primero debe
calcular la treliza.")
                return

            if not self.selected_tension_bars:
                messagebox.showwarning("Advertencia", "Debe seleccionar al
menos una barra traccionada.")
                return

            fy = self.fy.get()
            fR = self.fR.get()

            # Encabezado con información del proyecto
            results = f"=== DATOS DEL PROYECTO ===\n"
            results += f"Fecha: {self.project_info.get('date', 'N/A')}\n"
            results += f"Usuario: {self.project_info.get('user_name',
'N/A')}\n"
            results += f"Proyecto: {self.project_info.get('project_name',
'N/A')}\n\n"

            results += "=== CÁLCULO DE ARMADURAS PARA BARRAS TRACCIONADAS
===\n"
            results += f"fy: {fy} ksi\n"
            results += "Fórmula: As = Fuerza / (Φ * fy)\n\n"

            for bar_idx in self.selected_tension_bars:
                if bar_idx not in self.bar_forces:
                    continue

                fuerza = self.bar_forces[bar_idx]

                if fuerza <= 0:
                    results += f"Barra {bar_idx}: No está traccionada
(fuerza = {fuerza:.2f} kip)\n"
                    continue

                # Cálculo del área de acero según la fórmula As = Fuerza /
(0.75 * fy)
                As = fuerza / (fR * fy)
                results += f"Barra {bar_idx}: As = {fuerza:.2f} / (Φ ×
{fy:.2f}) = {As:.4f} in²\n"
```

```
            self.additional_results_text.delete(1.0, tk.END)
            self.additional_results_text.insert(tk.END, results)

        except Exception as e:
            messagebox.showerror("Error", f"Ocurrió un error: {str(e)}")

    def verify_struts(self):
        try:
            if not self.bar_forces:
                messagebox.showwarning("Advertencia", "Primero debe
calcular la treliza.")
                return

            if self.selected_compression_bar is None:
                messagebox.showwarning("Advertencia", "Debe seleccionar una
barra comprimida.")
                return

            # Obtener parámetros
            fc = self.fc.get()
            concrete_type = self.concrete_type_var.get()
            beta_s_type = self.beta_s_var.get()
            b = self.b_var.get()
            bs = self.bs_var.get()
            Bc = abs(self.bar_forces[self.selected_compression_bar])  #
Valor absoluto de la fuerza compresiva
            fR = self.fR.get()

            # Determinar λ según tipo de hormigón
            if concrete_type == "Hormigón de peso normal":
                lambda_val = 1.0
            elif concrete_type == "Hormigón ligero con arena":
                lambda_val = 0.85
            elif concrete_type == "Otro tipo de Hormigón ligero":
                lambda_val = 0.75
            else:
                lambda_val = 1.0  # Valor por defecto

            # Determinar βs según selección
            if beta_s_type == "Área de sección transversal uniforme":
                beta_s = 1.0
            elif beta_s_type == "Formato de botella con armadura":
                beta_s = 0.75
            elif beta_s_type == "Formato de botella sin armadura":
                beta_s = 0.6 * lambda_val
            elif beta_s_type == "Bielas en zonas de tensión":
                beta_s = 0.4
            elif beta_s_type == "Otros":
                beta_s = 0.6
            else:
                beta_s = 1.0  # Valor por defecto

            # Calcular fcu = 0.85 * βs * fc
            fcu = 0.85 * beta_s * fc

            # Calcular Ac_s = b * bs
```

```
            Ac_s = b * bs

            # Calcular fuerza resistente Fr = fcu * Ac_s
            Fr = fcu * Ac_s * fR

            # Verificación
            verification_results = "\n=== VERIFICACIÓN DE BIELAS
COMPRIMIDAS SEGÚN PCI ===\n\n"
            verification_results += f"Barra comprimida seleccionada:
{self.selected_compression_bar}\n"
            verification_results += f"Fuerza compresiva (Bc): {Bc:.2f}
kip\n"
            verification_results += f"Tipo de hormigón: {concrete_type}
(λ={lambda_val})\n"
            verification_results += f"Coeficiente de la biela:
{beta_s_type} (βs={beta_s:.2f})\n"
            verification_results += f"Resistencia a compresión del hormigón
(fc): {fc} ksi\n"
            verification_results += f"Resistencia efectiva de la biela
(fcu): 0.85*βs*fc = {fcu:.2f} ksi\n"
            verification_results += f"Área de la biela (Ac_s): b*bs =
{b:.2f}*{bs:.2f} = {Ac_s:.2f} in²\n"
            verification_results += f"Fuerza resistente (Fr): fcu*Ac_s =
{fcu:.2f}*{Ac_s:.2f}*Φ = {Fr:.2f} kip\n\n"

            verification_results += "=== RESULTADO DE LA VERIFICACIÓN
===\n"
            if Bc <= Fr:
                verification_results += f"Bc ({Bc:.2f} kip) ≤ Fr ({Fr:.2f}
kip) → LA BIELA RESISTE\n"
            else:
                verification_results += f"Bc ({Bc:.2f} kip) > Fr ({Fr:.2f}
kip) → LA BIELA NO RESISTE\n"

            self.additional_results_text.insert(tk.END,
verification_results)

        except Exception as e:
            messagebox.showerror("Error en Verificación", f"Ocurrió un
error: {e}")

    def plot_truss(self):
        self.ax.clear()

        # Dibujar nodos
        node_coords = np.array(self.nodes)
        self.ax.plot(node_coords[:, 0], node_coords[:, 1], 'o',
markersize=8, color='blue')
        for i, (x, y) in enumerate(self.nodes):
            self.ax.text(x, y, f' {i}', verticalalignment='bottom',
horizontalalignment='right')

        # Dibujar barras y fuerzas axiales
        for i, (n1, n2) in enumerate(self.bars):
            x1, y1 = self.nodes[n1]
            x2, y2 = self.nodes[n2]
```

```
            color = 'gray'
            linestyle = '-'
            if i in self.bar_forces:
                force = self.bar_forces[i]
                if force > 0: # Tensión
                    color = 'red'
                elif force < 0: # Compresión
                    color = 'green'

            self.ax.plot([x1, x2], [y1, y2], color=color,
linestyle=linestyle, linewidth=2)

            # Mostrar valor de la fuerza en la barra
            mid_x = (x1 + x2) / 2
            mid_y = (y1 + y2) / 2
            if i in self.bar_forces:
                self.ax.text(mid_x, mid_y, f'{self.bar_forces[i]:.2f} kip',
color='black',
                             fontsize=8, ha='center', va='center',
                             bbox=dict(facecolor='white', alpha=0.7,
edgecolor='none', boxstyle='round,pad=0.2'))

        # Dibujar fuerzas aplicadas
        for node_idx, (fx, fy) in self.forces.items():
            x, y = self.nodes[node_idx]
            arrow_scale = 0.5 # Escala para la longitud de la flecha

            # Dibujar flecha para Fx
            if fx != 0:
                self.ax.arrow(x, y, fx * arrow_scale / abs(fx) if fx != 0
else 0, 0,
                              head_width=0.1, head_length=0.1, fc='purple',
ec='purple')
                self.ax.text(x + (fx * arrow_scale / abs(fx) if fx != 0
else 0), y, f'Fx={fx}', color='purple', fontsize=8)

            # Dibujar flecha para Fy
            if fy != 0:
                self.ax.arrow(x, y, 0, fy * arrow_scale / abs(fy) if fy !=
0 else 0,
                              head_width=0.1, head_length=0.1, fc='orange',
ec='orange')
                self.ax.text(x, y + (fy * arrow_scale / abs(fy) if fy != 0
else 0), f'Fy={fy}', color='orange', fontsize=8)

        # Dibujar apoyos con los nuevos símbolos
        for node_idx, support_type in self.supports.items():
            x, y = self.nodes[node_idx]
            if support_type == 'fijo':
                # Cuadrado para apoyo fijo
                self.ax.plot(x, y - 0.2, 's', markersize=10, color='black',
markeredgewidth=1.5, markerfacecolor='none')
            elif support_type == 'movil_x':
                # Triángulo hacia arriba para móvil en X (restricción en Y)
                self.ax.plot(x, y - 0.2, '^', markersize=10, color='black',
markeredgewidth=1.5, markerfacecolor='none')
            elif support_type == 'movil_y':
```

```
                # Triángulo hacia la derecha para móvil en Y (restricción
en X)
                self.ax.plot(x - 0.2, y, '>', markersize=10, color='black',
markeredgewidth=1.5, markerfacecolor='none')
            elif support_type == 'empotrado':
                # Rombo para empotrado
                self.ax.plot(x, y - 0.2, 'D', markersize=10, color='black',
markeredgewidth=1.5, markerfacecolor='none')

        self.ax.set_xlabel("X (in)")
        self.ax.set_ylabel("Y (in)")
        self.ax.set_title("Visualización de la Treliza")
        self.ax.grid(True)
        self.ax.set_aspect('equal', adjustable='box')
        self.fig.tight_layout()
        self.canvas.draw()

class PCICantileverGUI:
    def __init__(self, root, project_info=None):
        self.root = root
        self.root.title("PCI - Viga en Balance")
        self.root.geometry("800x700")
        self.project_info = project_info if project_info else {}

        self.create_widgets()

    def create_widgets(self):
        canvas = tk.Canvas(self.root)
        scrollbar = ttk.Scrollbar(self.root, orient="vertical",
command=canvas.yview)
        self.scrollable_frame = ttk.Frame(canvas)

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0), window=self.scrollable_frame,
anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        # Parámetros Geométricos
        geo_frame = ttk.LabelFrame(self.scrollable_frame, text="Parámetros
Geométricos", padding=10)
        geo_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(geo_frame, text="Largura de la pieza (l)
[in]:").grid(row=0, column=0, sticky=tk.W, padx=5, pady=2)
        self.l_var = tk.DoubleVar()
        ttk.Entry(geo_frame, textvariable=self.l_var, width=15).grid(row=0,
column=1, padx=5, pady=2)

        ttk.Label(geo_frame, text="Altura efectiva (d) [in]:").grid(row=1,
column=0, sticky=tk.W, padx=5, pady=2)
```

```
        self.d_var = tk.DoubleVar()
        ttk.Entry(geo_frame, textvariable=self.d_var, width=15).grid(row=1,
column=1, padx=5, pady=2)

        ttk.Label(geo_frame, text="Altura de la pieza (h)
[in]:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        self.h_var = tk.DoubleVar()
        ttk.Entry(geo_frame, textvariable=self.h_var, width=15).grid(row=2,
column=1, padx=5, pady=2)

        ttk.Label(geo_frame, text="Ancho de la pieza (b)
[in]:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=2)
        self.b_var = tk.DoubleVar()
        ttk.Entry(geo_frame, textvariable=self.b_var, width=15).grid(row=3,
column=1, padx=5, pady=2)

        # Factores de Seguridad y Materiales
        factors_frame = ttk.LabelFrame(self.scrollable_frame,
text="Factores de Seguridad y Materiales", padding=10)
        factors_frame.pack(fill=tk.X, padx=5, pady=5)

        # Selección de tipo de hormigón (λ)
        ttk.Label(factors_frame, text="Tipo de hormigón (λ):").grid(row=0,
column=0, sticky=tk.W, padx=5, pady=2)
        self.concrete_type_var = tk.StringVar(value="Hormigón de peso
normal")
        concrete_type_frame = ttk.Frame(factors_frame)
        concrete_type_frame.grid(row=0, column=1, columnspan=2,
sticky=tk.W, padx=5, pady=2)
        ttk.Radiobutton(concrete_type_frame, text="Hormigón de peso normal
(λ=1)",
                        variable=self.concrete_type_var, value="Hormigón de
peso normal").pack(anchor=tk.W)
        ttk.Radiobutton(concrete_type_frame, text="Hormigón ligero con
arena (λ=0.85)",
                        variable=self.concrete_type_var, value="Hormigón
ligero con arena").pack(anchor=tk.W)
        ttk.Radiobutton(concrete_type_frame, text="Otro tipo de Hormigón
ligero (λ=0.75)",
                        variable=self.concrete_type_var, value="Otro tipo de
Hormigón ligero").pack(anchor=tk.W)

        # Selección de condición de interface (μ)
        ttk.Label(factors_frame, text="Condición de la interface
(μ):").grid(row=1, column=0, sticky=tk.W, padx=5, pady=2)
        self.interface_condition_var = tk.StringVar(value="Hormigón lanzado
Monoliticamente")
        interface_frame = ttk.Frame(factors_frame)
        interface_frame.grid(row=1, column=1, columnspan=2, sticky=tk.W,
padx=5, pady=2)
        ttk.Radiobutton(interface_frame, text="Hormigón lanzado
Monoliticamente (μ=1.4λ, μmax=3.4)",
                        variable=self.interface_condition_var,
value="Hormigón lanzado Monoliticamente").pack(anchor=tk.W)
        ttk.Radiobutton(interface_frame, text="En 2 etapas con superficie
rugosa (μ=1λ, μmax=2.9)",
```

```
                        variable=self.interface_condition_var, value="En 2
etapas con superficie rugosa").pack(anchor=tk.W)
        ttk.Radiobutton(interface_frame, text="En 2 etapas sin superficie
rugosa (μ=0.6λ, μmax=N/A)",
                        variable=self.interface_condition_var, value="En 2
etapas sin superficie rugosa").pack(anchor=tk.W)
        ttk.Radiobutton(interface_frame, text="Apoyado sobre chapa de acero
(μ=0.7λ, μmax=N/A)",
                        variable=self.interface_condition_var,
value="Apoyado sobre chapa de acero").pack(anchor=tk.W)

        ttk.Label(factors_frame, text="Factor reductor de resistencia
(Φ):").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        self.phi_var = tk.DoubleVar(value=0.75)
        ttk.Entry(factors_frame, textvariable=self.phi_var,
width=15).grid(row=2, column=1, padx=5, pady=2)

        ttk.Label(factors_frame, text="Resistencia a fluencia del acero
(fy) [ksi]:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=2)
        self.fy_var = tk.DoubleVar(value=60)  # 500 MPa = 72.5 ksi
        ttk.Entry(factors_frame, textvariable=self.fy_var,
width=15).grid(row=3, column=1, padx=5, pady=2)

        ttk.Label(factors_frame, text="Resistencia a compresión del
concreto (fc) [ksi]:").grid(row=4, column=0, sticky=tk.W, padx=5, pady=2)
        self.fc_var = tk.DoubleVar(value=5)  # 25 MPa = 3.625 ksi
        ttk.Entry(factors_frame, textvariable=self.fc_var,
width=15).grid(row=4, column=1, padx=5, pady=2)

        # Fuerzas Aplicadas
        forces_frame = ttk.LabelFrame(self.scrollable_frame, text="Cargas
Aplicadas", padding=10)
        forces_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(forces_frame, text="Carga Vertical (V)
[kip]:").grid(row=0, column=0, sticky=tk.W, padx=5, pady=2)
        self.V_var = tk.DoubleVar()
        ttk.Entry(forces_frame, textvariable=self.V_var,
width=15).grid(row=0, column=1, padx=5, pady=2)

        ttk.Label(forces_frame, text="Carga Horizontal (H)
[kip]:").grid(row=1, column=0, sticky=tk.W, padx=5, pady=2)
        self.H_var = tk.DoubleVar()
        ttk.Entry(forces_frame, textvariable=self.H_var,
width=15).grid(row=1, column=1, padx=5, pady=2)

        # Botón de cálculo
        calc_button = ttk.Button(self.scrollable_frame, text="CALCULAR",
command=self.calculate_cantilever, style="Accent.TButton")
        calc_button.pack(pady=10)

        # Cuadro de resultados
        results_frame = ttk.LabelFrame(self.scrollable_frame,
text="Resultados", padding=10)
        results_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)
```

```
        self.results_text = scrolledtext.ScrolledText(results_frame,
height=40, width=80)
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def get_lambda_value(self):
        """Obtiene el valor de λ según el tipo de hormigón seleccionado"""
        concrete_type = self.concrete_type_var.get()
        if concrete_type == "Hormigón de peso normal":
            return 1.0
        elif concrete_type == "Hormigón ligero con arena":
            return 0.85
        elif concrete_type == "Otro tipo de Hormigón ligero":
            return 0.75
        return 1.0  # Valor por defecto

    def get_mu_values(self):
        """Obtiene μ y μmax según la condición de interface seleccionada"""
        interface_condition = self.interface_condition_var.get()
        lambda_val = self.get_lambda_value()

        if interface_condition == "Hormigón lanzado Monoliticamente":
            return 1.4 * lambda_val, 3.4
        elif interface_condition == "En 2 etapas con superficie rugosa":
            return 1.0 * lambda_val, 2.9
        elif interface_condition == "En 2 etapas sin superficie rugosa":
            return 0.6 * lambda_val, None  # μmax no aplicable
        elif interface_condition == "Apoyado sobre chapa de acero":
            return 0.7 * lambda_val, None  # μmax no aplicable
        return 1.4 * lambda_val, 3.4  # Valores por defecto

    def calculate_cantilever(self):
        try:
            l = self.l_var.get()
            d = self.d_var.get()
            h = self.h_var.get()
            b = self.b_var.get()
            phi = self.phi_var.get()
            fy = self.fy_var.get()
            fc = self.fc_var.get()
            V = self.V_var.get()
            H = self.H_var.get()

            # Obtener λ según tipo de hormigón
            lambda_val = self.get_lambda_value()

            # Obtener μ y μmax según condición de interface
            mu, mu_max = self.get_mu_values()
            interface_condition = self.interface_condition_var.get()

            results_output = ""
            results_output += f"Fecha: {self.project_info.get('date',
'N/A')}\n"
            results_output += f"Nombre del Proyecto:
{self.project_info.get('project_name', 'N/A')}\n"
            results_output += f"Nombre del Usuario:
{self.project_info.get('user_name', 'N/A')}\n\n"
```

```
            results_output += "=== Parámetros de Entrada ===\n"
            results_output += f"l: {l} in\n"
            results_output += f"d: {d} in\n"
            results_output += f"h: {h} in\n"
            results_output += f"b: {b} in\n"
            results_output += f"Tipo de hormigón:
{self.concrete_type_var.get()} (λ={lambda_val})\n"
            results_output += f"Condición de interface:
{self.interface_condition_var.get()} (μ={mu:.2f}"
            if mu_max is not None:
                results_output += f", μmax={mu_max}"
            results_output += ")\n"
            results_output += f"Φ: {phi}\n"
            results_output += f"fy: {fy} ksi\n"
            results_output += f"fc: {fc} ksi\n"
            results_output += f"V: {V} kip\n"
            results_output += f"H: {H} kip\n\n"

            # 1. Verifica que a/d <= 1
            a = (3/4)*l
            results_output += f"Punto de aplicación de la carga (a) = 3/4*l
= {a:.2f} in\n"
            if d == 0:
                results_output += "Error: 'd' no puede ser cero para la
verificación a/d.\n"
                self.results_text.delete(1.0, tk.END)
                self.results_text.insert(tk.END, results_output)
                return

            a_d_ratio = a / d
            results_output += f"1. Verificación a/d <= 1 (a/d =
{a_d_ratio:.2f}): "
            if a_d_ratio > 1:
                results_output += "Viola la condición de utilización del
método\n"
            else:
                results_output += "OK\n"

            # 2. Verifica que H <= V
            results_output += f"2. Verificación H <= V (H={H} kip, V={V}
kip): "
            if H > V:
                results_output += "Viola la condición de utilización del
método\n"
            else:
                results_output += "OK\n"

            # 3. Acr = b * h
            Acr = b * h

            results_output += f"3. Área de la interface fisurada (Acr) = b
* h = {Acr:.2f} in²\n"
            # 4. V_max = (Φ * 1000 * λ * Acr) / 1000
            if interface_condition == "Hormigón lanzado Monoliticamente":
                V_max = (phi * 1000 * lambda_val * Acr) / 1000
                results_output += f"4. Carga vertical máxima (V_max) = (Φ *
1000 * λ * Acr) / 1000 = {V_max:.2f} kip\n"
```

```
            elif interface_condition == "En 2 etapas con superficie
rugosa":
                V_max = (phi * 1000 * lambda_val * Acr) / 1000
                results_output += f"4. Carga vertical máxima (V_max) = (Φ *
1000 * λ * Acr) / 1000 = {V_max:.2f} kip\n"
            elif interface_condition == "En 2 etapas sin superficie
rugosa":
                V_max = (phi * 800 * lambda_val * Acr) / 1000
                results_output += f"4. Carga vertical máxima (V_max) = (Φ *
800 * λ * Acr) / 1000 = {V_max:.2f} kip\n"
            elif interface_condition == "Apoyado sobre chapa de acero":
                V_max = (phi * 800 * lambda_val * Acr) / 1000
                results_output += f"4. Carga vertical máxima (V_max) = (Φ *
800 * λ * Acr) / 1000 = {V_max:.2f} kip\n"

            # 5. V_max > V
            results_output += f"5. Verificación V_max > V
(V_max={V_max:.2f} kip, V={V} kip): "
            if V_max <= V:
                results_output += "Viola la condición de utilización del
método\n"
            else:
                results_output += "OK\n"

            # 6. As_t1 = (1/Φ*fy)*((V*(a/d))+(H*(h/d)))
            As_t1 = (1 / (phi * fy)) * ((V * (a / d)) + (H * (h / d)))
            results_output += f"6. As_t1 = (1/Φ*fy)*((V*(a/d))+(H*(h/d))) =
{As_t1:.4f} in²\n"

            # 7. As_t2=(1/Φ*fy)*((2*V/3 μe)+H)

            # Calcular μe = (Φ*1000*λ*Acr*μ)/V
            mu_e = (phi * 1000 * lambda_val * Acr * mu) / (V*1000) if V !=
0 else 0
            results_output += f"7. μe = (Φ*1000*λ*Acr*μ)/(V*1000) =
{mu_e:.4f}\n"

            if mu_max is not None:
                results_output += f"   Verificación μe <= μmax ({mu_e:.4f}
<= {mu_max}): "
                if mu_e <= mu_max:
                    results_output += "OK\n"
                else:
                    results_output += "NO CUMPLE\n"
            else:
                results_output += "   μmax no aplicable para esta condición
de interface\n"

            As_t2 = (1 / (phi * fy)) * (((2 * V) / (3 * mu_e)) + H) if mu_e
!= 0 else 0
            results_output += f"8. As_t2=(1/Φ*fy)*((2*V/3 μe)+H) =
{As_t2:.4f} in²\n"

            # 9. As_t = Mayor entre (Ast_1;As_t2)
            As_t = max(As_t1, As_t2)
            results_output += f"9. Armadura del tirante (As_t) = (mayor
entre As_t1 y As_t2) = {As_t:.4f} in²\n"
```

```
            # 10. As_min= 0,04*(fc/fy)*b*d
            As_min = 0.04 * (fc / fy) * b * d
            results_output += f"10. Armadura de tirante mínima (As_min) =
0,04*(fc/fy)*b*d = {As_min:.4f} in²\n"

            # 11. As_min <= As_t
            As_td = As_t # Valor por defecto
            results_output += f"11. Verificación As_min <= As_t
(As_min={As_min:.4f} in², As_t={As_t:.4f} in²): "
            if As_min > As_t:
                results_output += "Armadura mínima requerida. "
                if messagebox.askyesno("Armadura Mínima",
                                       f"As_min ({As_min:.4f} in²) es mayor
que As_t ({As_t:.4f} in²).\n" +
                                       "¿Desea usar As_min como As_td?"):
                    As_td = As_min
                    results_output += f"Se ha elegido As_min como As_td
({As_td:.4f} in²).\n"
                else:
                    As_td = As_t
                    results_output += f"Se ha elegido As_t como As_td
({As_td:.4f} in²).\n"
            else:
                results_output += "OK\n"
                results_output += f"   As_td: {As_td:.4f} in²\n"

            # 12. An=H/Φ*fy
            An = H / (phi * fy)
            results_output += f"12. An = H/Φ*fy = {An:.4f} in²\n"

            # 13. As_c=0,5*((As_td)-An)
            As_c = 0.5 * (As_td - An)
            results_output += f"13. Ármadura de costura (As_c) =
0,5*((As_td)-An) = {As_c:.4f} in²\n Obs.: Segundo o modelo As_c <
0,5*((As_td)-An)"

            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, results_output)

        except ValueError:
            messagebox.showerror("Error de Entrada", "Por favor, ingrese
valores numéricos válidos en todos los campos.")
        except Exception as e:
            messagebox.showerror("Error de Cálculo", f"Ocurrió un error
durante el cálculo: {e}")

class NBRConsolosCurtosGUI:
    def __init__(self, root, project_info=None):
        self.root = root
        self.root.title("NBR - Ménsulas Cortas")
        self.root.geometry("800x700")
        self.project_info = project_info if project_info else {}

        # Dictionary of support types and their γa values
        self.support_types = {
            "Juntas a seco": 0.8,
```

```
            "Asentado con masa": 0.5,
            "Elastómero": 0.16,
            "Plástico Politetrafluoretileno (PTFE)": 0.08,
            "Chapas metálicas no soldadas": 0.25,
            "Hormigón con Chapa": 0.4,
            "Solda com apoyo ou engrudado": 0.3,
            "Otro, especifique": None
        }

        self.gamma_a_var = tk.DoubleVar(value=0.5)  # Default value
        self.custom_gamma_a_var = tk.DoubleVar()
        self.support_type_var = tk.StringVar(value="Asentado con masa")  #
Default selection
        self.auto_calculate_H = tk.BooleanVar(value=True)  # Auto-calculate
H by default

        self.create_widgets()

    def create_widgets(self):
        canvas = tk.Canvas(self.root)
        scrollbar = ttk.Scrollbar(self.root, orient="vertical",
command=canvas.yview)
        self.scrollable_frame = ttk.Frame(canvas)

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0), window=self.scrollable_frame,
anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        # Parámetros de Entrada
        input_frame = ttk.LabelFrame(self.scrollable_frame, text="Factores
de seguridad", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(input_frame, text="Tipo de Carga:").grid(row=0, column=0,
sticky=tk.W, padx=5, pady=2)
        self.load_type_var = tk.StringVar(value="Directa")
        ttk.Radiobutton(input_frame, text="Directa",
variable=self.load_type_var, value="Directa").grid(row=0, column=1,
sticky=tk.W, padx=5, pady=2)
        ttk.Radiobutton(input_frame, text="Indirecta",
variable=self.load_type_var, value="Indirecta").grid(row=0, column=2,
sticky=tk.W, padx=5, pady=2)

        # Support type selection
        ttk.Label(input_frame, text="Tipo de Apoyo γa:").grid(row=1,
column=0, sticky=tk.W, padx=5, pady=2)
        support_combo = ttk.Combobox(input_frame,
textvariable=self.support_type_var,
```

```
                                    values=list(self.support_types.keys()),
width=30)
        support_combo.grid(row=1, column=1, columnspan=2, sticky=tk.W,
padx=5, pady=2)
        support_combo.bind("<<ComboboxSelected>>", self.update_gamma_a)

        # Custom gamma_a entry (hidden by default)
        self.custom_gamma_frame = ttk.Frame(input_frame)
        ttk.Label(self.custom_gamma_frame, text="γa
personalizado:").pack(side=tk.LEFT, padx=5)
        ttk.Entry(self.custom_gamma_frame,
textvariable=self.custom_gamma_a_var, width=10).pack(side=tk.LEFT)

        # Auto-calculate H checkbox
        self.auto_calc_frame = ttk.Frame(input_frame)
        self.auto_calc_frame.grid(row=2, column=0, columnspan=3,
sticky=tk.W, padx=5, pady=2)
        ttk.Checkbutton(self.auto_calc_frame, text="Calcular H
automáticamente (H = V × γa)",
                        variable=self.auto_calculate_H,
command=self.toggle_H_calculation).pack(side=tk.LEFT)

        input_frame = ttk.LabelFrame(self.scrollable_frame,
text="Parámetros Geométricos", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(input_frame, text="Punto de aplicación de carga V (a)
[mm]:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=2)
        self.a_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.a_var,
width=15).grid(row=3, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Altura efectiva (d)
[mm]:").grid(row=4, column=0, sticky=tk.W, padx=5, pady=2)
        self.d_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.d_var,
width=15).grid(row=4, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Ancho de la pieza (b)
[mm]:").grid(row=5, column=0, sticky=tk.W, padx=5, pady=2)
        self.b_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.b_var,
width=15).grid(row=5, column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(self.scrollable_frame,
text="Materiales", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(input_frame, text="Resistencia a fluencia del acero (fy)
[MPa]:").grid(row=6, column=0, sticky=tk.W, padx=5, pady=2)
        self.fy_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.fy_var,
width=15).grid(row=6, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Resistencia a compresión del concreto
(fc) [MPa]:").grid(row=7, column=0, sticky=tk.W, padx=5, pady=2)
        self.fc_var = tk.DoubleVar()
```

```
        ttk.Entry(input_frame, textvariable=self.fc_var,
width=15).grid(row=7, column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(self.scrollable_frame, text="Cargas y
Tensión Actuantes ", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)


        ttk.Label(input_frame, text="Carga vertical (V) [N]:").grid(row=8,
column=0, sticky=tk.W, padx=5, pady=2)
        self.V_var = tk.DoubleVar()
        self.V_entry = ttk.Entry(input_frame, textvariable=self.V_var,
width=15)
        self.V_entry.grid(row=8, column=1, padx=5, pady=2)
        self.V_var.trace_add("write", self.calculate_H_if_auto)

        ttk.Label(input_frame, text="Carga Horizontal (H)
[N]:").grid(row=9, column=0, sticky=tk.W, padx=5, pady=2)
        self.H_var = tk.DoubleVar()
        self.H_entry = ttk.Entry(input_frame, textvariable=self.H_var,
width=15, state='disabled')
        self.H_entry.grid(row=9, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Tensión de compresión actuante (σ)
[N/mm²]:").grid(row=10, column=0, sticky=tk.W, padx=5, pady=2)
        self.tau_w_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.tau_w_var,
width=15).grid(row=10, column=1, padx=5, pady=2)

        # Botón de cálculo
        calc_button = ttk.Button(self.scrollable_frame, text="CALCULAR",
command=self.calculate_nbr_cantilever, style="Accent.TButton")
        calc_button.pack(pady=10)

        # Cuadro de resultados
        results_frame = ttk.LabelFrame(self.scrollable_frame,
text="Resultados", padding=10)
        results_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.results_text = scrolledtext.ScrolledText(results_frame,
height=40, width=90)
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def update_gamma_a(self, event=None):
        """Update gamma_a based on selected support type"""
        selected_type = self.support_type_var.get()

        # Remove custom gamma_a frame if it exists
        if hasattr(self, 'custom_gamma_frame'):
            self.custom_gamma_frame.grid_forget()

        if selected_type == "Otro, especifique":
            # Show custom gamma_a entry
            self.custom_gamma_frame.grid(row=1, column=3, sticky=tk.W,
padx=5, pady=2)
        else:
            # Set gamma_a from predefined values
```

```
            self.gamma_a_var.set(self.support_types[selected_type])

        # Recalculate H if auto-calculate is enabled
        self.calculate_H_if_auto()

    def toggle_H_calculation(self):
        """Toggle automatic H calculation"""
        if self.auto_calculate_H.get():
            self.H_entry.config(state='disabled')
            self.calculate_H_if_auto()
        else:
            self.H_entry.config(state='normal')

    def calculate_H_if_auto(self, *args):
        """Calculate H = V * γa if auto-calculate is enabled"""
        if self.auto_calculate_H.get():
            try:
                V = float(self.V_var.get())
                gamma_a = self.get_current_gamma_a()
                H = V * gamma_a
                self.H_var.set(round(H, 2))
            except (ValueError, TypeError):
                # Ignore if values aren't valid numbers yet
                pass

    def get_current_gamma_a(self):
        """Get the current gamma_a value based on selection"""
        if self.support_type_var.get() == "Otro, especifique":
            try:
                return float(self.custom_gamma_a_var.get())
            except (ValueError, TypeError):
                return 0.0
        else:
            return self.support_types[self.support_type_var.get()]

    def calculate_nbr_cantilever(self):
        try:
            load_type = self.load_type_var.get()
            support_type = self.support_type_var.get()
            gamma_a = self.get_current_gamma_a()

            # Get V and H values
            V = self.V_var.get()
            if self.auto_calculate_H.get():
                H = V * gamma_a
                self.H_var.set(round(H, 2))  # Update displayed H value
            else:
                H = self.H_var.get()

            a = self.a_var.get()
            d = self.d_var.get()
            b = self.b_var.get()
            fy = self.fy_var.get()
            fc = self.fc_var.get()
            tau_w = self.tau_w_var.get()

            results_output = ""
```

```
            results_output += f"Fecha: {self.project_info.get('date',
'N/A')}\n"
            results_output += f"Nombre del Proyecto:
{self.project_info.get('project_name', 'N/A')}\n"
            results_output += f"Nombre del Usuario:
{self.project_info.get('user_name', 'N/A')}\n\n"

            results_output += "=== Parámetros de Entrada ===\n"
            results_output += f"Tipo de Carga: {load_type}\n"
            results_output += f"Tipo de Apoyo: {support_type}
(γa={gamma_a})\n"
            results_output += f"V: {V} N\n"
            if self.auto_calculate_H.get():
                results_output += f"H (calculado automáticamente): {H:.2f}
N (H = V × γa = {V} × {gamma_a})\n"
            else:
                results_output += f"H (ingresado manualmente): {H} N\n"
            results_output += f"a: {a} mm\n"
            results_output += f"d: {d} mm\n"
            results_output += f"b: {b} mm\n"
            results_output += f"fy: {fy} MPa\n"
            results_output += f"fc: {fc} MPa\n"
            results_output += f"σ: {tau_w} N/mm²\n\n"

            results_output += "=== Resultados ===\n"
            # 1. Verifica que 0.5 < a/d <= 1
            if d == 0:
                results_output += "Error: 'd' no puede ser cero para la
verificación a/d.\n"
                self.results_text.delete(1.0, tk.END)
                self.results_text.insert(tk.END, results_output)
                return

            a_d_ratio = a / d
            results_output += f"1. Verificación 0.5 < a/d <= 1 (a/d =
{a_d_ratio:.2f}): "
            if not (0.5 < a_d_ratio <= 1):
                results_output += "Viola la condición de utilización del
método\n"
            else:
                results_output += "OK\n"

            # 2. Asv = (0.1 + (a/d)) * (V/fy)
            Asv = (0.1 + (a / d)) * (V / fy)
            results_output += f"2. Armadura resistente a carga vertical
(Asv) = (0.1 + (a/d)) * (V/fy) = {Asv:.4f} mm²\n"

            # 3. As_t = Asv + (H/fy)
            As_t = Asv + (H / fy)
            results_output += f"3. Armadura del tirante (As_t) = Asv +
(H/fy) = {As_t:.4f} mm²\n"

            # 4. As_c = 0.4 * (Asv/d)
            As_c = 0.4 * (Asv / d)
            results_output += f"4. Armadura de costura (As_c) = 0.4 *
(Asv/d) {As_c:.4f} mm²\n"
```

```
            # 5. Verificación según tipo de carga
            results_output += f"\n=== Verificación de la biela de
compresión ===\n"

            if load_type == "Directa":
                results_output += f"5. Condición para carga directa: σ <=
fc\n"
                results_output += f"σ = {tau_w:.2f} N/mm²\n"
                results_output += f"fc = {fc:.2f} N/mm²\n"
                if tau_w <= fc:
                    results_output += "OK - La biela resiste (σ <= fc)\n"
                else:
                    results_output += "NO CUMPLE - La biela no resiste (σ >
fc)\n"
            else:  # Carga indirecta
                tau_w_limit = 0.85 * fc
                results_output += f"5. Condición para carga indirecta: σ <=
0.85*fc\n"
                results_output += f"σ = {tau_w:.2f} N/mm²\n"
                results_output += f"0.85*fc = {tau_w_limit:.2f} N/mm²\n"
                if tau_w <= tau_w_limit:
                    results_output += "OK - La biela resiste (σ <=
0.85*fc)\n"
                else:
                    results_output += "NO CUMPLE - La biela no resiste (σ >
0.85*fc)\n"

            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, results_output)

        except ValueError as ve:
            messagebox.showerror("Error de Entrada", str(ve))
        except Exception as e:
            messagebox.showerror("Error de Cálculo", f"Ocurrió un error
durante el cálculo: {e}")

class NBRConsolosMuitoCurtosGUI:
    def __init__(self, root, project_info=None):
        self.root = root
        self.root.title("NBR - Ménsulas Muy Cortas")
        self.root.geometry("800x700")
        self.project_info = project_info if project_info else {}

        # Dictionary of support types and their γa values
        self.support_types = {
            "Juntas a seco": 0.8,
            "Asentado con masa": 0.5,
            "Elastómero": 0.16,
            "Plástico Politetrafluoretileno (PTFE)": 0.08,
            "Chapas metálicas no soldadas": 0.25,
            "Hormigón con Chapa": 0.4,
            "Solda com apoio ou engrudado": 0.3,
            "Otro, especifique": None
        }

        self.gamma_a_var = tk.DoubleVar(value=0.5)  # Default value
        self.custom_gamma_a_var = tk.DoubleVar()
```

```
        self.support_type_var = tk.StringVar(value="Asentado con masa")  #
Default selection
        self.auto_calculate_H = tk.BooleanVar(value=True)  # Auto-calculate
H by default

        self.create_widgets()

    def create_widgets(self):
        canvas = tk.Canvas(self.root)
        scrollbar = ttk.Scrollbar(self.root, orient="vertical",
command=canvas.yview)
        self.scrollable_frame = ttk.Frame(canvas)

        self.scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0), window=self.scrollable_frame,
anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        # Parámetros de Entrada
        input_frame = ttk.LabelFrame(self.scrollable_frame, text="Factores
de seguridad", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        # Support type selection
        ttk.Label(input_frame, text="Tipo de Apoyo γa:").grid(row=0,
column=0, sticky=tk.W, padx=5, pady=2)
        support_combo = ttk.Combobox(input_frame,
textvariable=self.support_type_var,
                                     values=list(self.support_types.keys()),
width=30)
        support_combo.grid(row=0, column=1, columnspan=2, sticky=tk.W,
padx=5, pady=2)
        support_combo.bind("<<ComboboxSelected>>", self.update_gamma_a)

        # Custom gamma_a entry (hidden by default)
        self.custom_gamma_frame = ttk.Frame(input_frame)
        ttk.Label(self.custom_gamma_frame, text="γa
personalizado:").pack(side=tk.LEFT, padx=5)
        ttk.Entry(self.custom_gamma_frame,
textvariable=self.custom_gamma_a_var, width=10).pack(side=tk.LEFT)

        # Auto-calculate H checkbox
        self.auto_calc_frame = ttk.Frame(input_frame)
        self.auto_calc_frame.grid(row=1, column=0, columnspan=3,
sticky=tk.W, padx=5, pady=2)
        ttk.Checkbutton(self.auto_calc_frame, text="Calcular H
automáticamente (H = V × γa)",
                        variable=self.auto_calculate_H,
command=self.toggle_H_calculation).pack(side=tk.LEFT)
```

```
        # Coeficiente de rugosidad μ
        ttk.Label(input_frame, text="Coeficiente de rugosidad
μ:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        self.mu_var = tk.DoubleVar(value=1.4)
        mu_frame = ttk.Frame(input_frame)
        mu_frame.grid(row=2, column=1, columnspan=2, sticky=tk.W, padx=5,
pady=2)
        ttk.Radiobutton(mu_frame, text="1.4 (Hormigón lanzado
monolíticamente)", variable=self.mu_var, value=1.4).pack(anchor=tk.W)
        ttk.Radiobutton(mu_frame, text="1.0 (Hormigón sobre hormigón
endurecido)", variable=self.mu_var, value=1.0).pack(anchor=tk.W)
        ttk.Radiobutton(mu_frame, text="0.6 (Hormigón sobre hormigón
endurecido con interface lisa)", variable=self.mu_var,
value=0.6).pack(anchor=tk.W)

        input_frame = ttk.LabelFrame(self.scrollable_frame,
text="Parámetros Geométricos", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(input_frame, text="Punto de aplicación de la carga V (a)
[mm]:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=2)
        self.a_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.a_var,
width=15).grid(row=3, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Altura efectiva (d)
[mm]:").grid(row=4, column=0, sticky=tk.W, padx=5, pady=2)
        self.d_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.d_var,
width=15).grid(row=4, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Ancho de la pieza (b)
[mm]:").grid(row=5, column=0, sticky=tk.W, padx=5, pady=2)
        self.b_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.b_var,
width=15).grid(row=5, column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(self.scrollable_frame,
text="Materiales", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(input_frame, text="Resistencia a fluencia del acero (fy)
[MPa]:").grid(row=6, column=0, sticky=tk.W, padx=5, pady=2)
        self.fy_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.fy_var,
width=15).grid(row=6, column=1, padx=5, pady=2)
        # Added note about maximum fy value
        ttk.Label(input_frame, text="(fy ≤ 435 MPa - valor máximo tensión
máxima cizallante)",
                  font=('TkDefaultFont', 12)).grid(row=6, column=2,
sticky=tk.W, padx=5, pady=2)

        ttk.Label(input_frame, text="Resistencia a compresión del concreto
(fc) [MPa]:").grid(row=7, column=0, sticky=tk.W, padx=5, pady=2)
        self.fc_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.fc_var,
width=15).grid(row=7, column=1, padx=5, pady=2)
```

```
        ttk.Label(input_frame, text="Resistencia a compresión
característica del concreto (fck) [MPa]:").grid(row=8, column=0,
sticky=tk.W, padx=5, pady=2)
        self.fck_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.fck_var,
width=15).grid(row=8, column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(self.scrollable_frame, text="Cargas y
Tensión Actuantes", padding=10)
        input_frame.pack(fill=tk.X, padx=5, pady=5)

        ttk.Label(input_frame, text="Carga vertical (V) [N]:").grid(row=9,
column=0, sticky=tk.W, padx=5, pady=2)
        self.V_var = tk.DoubleVar()
        self.V_entry = ttk.Entry(input_frame, textvariable=self.V_var,
width=15)
        self.V_entry.grid(row=9, column=1, padx=5, pady=2)
        self.V_var.trace_add("write", self.calculate_H_if_auto)

        ttk.Label(input_frame, text="Carga Horizontal (H)
[N]:").grid(row=10, column=0, sticky=tk.W, padx=5, pady=2)
        self.H_var = tk.DoubleVar()
        self.H_entry = ttk.Entry(input_frame, textvariable=self.H_var,
width=15, state='disabled')
        self.H_entry.grid(row=10, column=1, padx=5, pady=2)

        # τw moved after H
        ttk.Label(input_frame, text="Tensión cizallante actuante (τw)
[MPa]:").grid(row=11, column=0, sticky=tk.W, padx=5, pady=2)
        self.tau_w_var = tk.DoubleVar()
        ttk.Entry(input_frame, textvariable=self.tau_w_var,
width=15).grid(row=11, column=1, padx=5, pady=2)

        # Botón de cálculo
        calc_button = ttk.Button(self.scrollable_frame, text="CALCULAR",
command=self.calculate_nbr_muito_curtos, style="Accent.TButton")
        calc_button.pack(pady=10)

        # Cuadro de resultados
        results_frame = ttk.LabelFrame(self.scrollable_frame,
text="Resultados", padding=10)
        results_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.results_text = scrolledtext.ScrolledText(results_frame,
height=40, width=90)
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    def update_gamma_a(self, event=None):
        """Update gamma_a based on selected support type"""
        selected_type = self.support_type_var.get()

        # Remove custom gamma_a frame if it exists
        if hasattr(self, 'custom_gamma_frame'):
            self.custom_gamma_frame.grid_forget()

        if selected_type == "Otro, especifique":
```

```
            # Show custom gamma_a entry
            self.custom_gamma_frame.grid(row=0, column=3, sticky=tk.W,
padx=5, pady=2)
        else:
            # Set gamma_a from predefined values
            self.gamma_a_var.set(self.support_types[selected_type])

        # Recalculate H if auto-calculate is enabled
        self.calculate_H_if_auto()

    def toggle_H_calculation(self):
        """Toggle automatic H calculation"""
        if self.auto_calculate_H.get():
            self.H_entry.config(state='disabled')
            self.calculate_H_if_auto()
        else:
            self.H_entry.config(state='normal')

    def calculate_H_if_auto(self, *args):
        """Calculate H = V * γa if auto-calculate is enabled"""
        if self.auto_calculate_H.get():
            try:
                V = float(self.V_var.get())
                gamma_a = self.get_current_gamma_a()
                H = V * gamma_a
                self.H_var.set(round(H, 2))
            except (ValueError, TypeError):
                # Ignore if values aren't valid numbers yet
                pass

    def get_current_gamma_a(self):
        """Get the current gamma_a value based on selection"""
        if self.support_type_var.get() == "Otro, especifique":
            try:
                return float(self.custom_gamma_a_var.get())
            except (ValueError, TypeError):
                return 0.0
        else:
            return self.support_types[self.support_type_var.get()]

    def calculate_nbr_muito_curtos(self):
        try:
            support_type = self.support_type_var.get()
            gamma_a = self.get_current_gamma_a()

            # Get V and H values
            V = self.V_var.get()
            if self.auto_calculate_H.get():
                H = V * gamma_a
                self.H_var.set(round(H, 2))  # Update displayed H value
            else:
                H = self.H_var.get()

            mu = self.mu_var.get()
            tau_w = self.tau_w_var.get()
            a = self.a_var.get()
            d = self.d_var.get()
```

```
            b = self.b_var.get()
            fy = self.fy_var.get()
            fc = self.fc_var.get()
            fck = self.fck_var.get()

            results_output = ""
            results_output += f"Fecha: {self.project_info.get('date',
'N/A')}\n"
            results_output += f"Nombre del Proyecto:
{self.project_info.get('project_name', 'N/A')}\n"
            results_output += f"Nombre del Usuario:
{self.project_info.get('user_name', 'N/A')}\n\n"

            results_output += "=== Parámetros de Entrada ===\n"
            results_output += f"Tipo de Apoyo: {support_type}
(γa={gamma_a})\n"
            results_output += f"V: {V} N\n"
            if self.auto_calculate_H.get():
                results_output += f"H (calculado automáticamente): {H:.2f}
N (H = V × γa = {V} × {gamma_a})\n"
            else:
                results_output += f"H (ingresado manualmente): {H} N\n"
            results_output += f"τw: {tau_w} MPa\n"
            results_output += f"μ: {mu}\n"
            results_output += f"a: {a} mm\n"
            results_output += f"d: {d} mm\n"
            results_output += f"b: {b} mm\n"
            results_output += f"fy: {fy} MPa (valor máximo tensión máxima
cizallante: 435 MPa)\n"
            results_output += f"fc: {fc} MPa\n"
            results_output += f"fck: {fck} MPa\n\n"

            # 1. Verifica que 0.5 => a/d
            if d == 0:
                results_output += "Error: 'd' no puede ser cero para la
verificación a/d.\n"
                self.results_text.delete(1.0, tk.END)
                self.results_text.insert(tk.END, results_output)
                return

            results_output += "=== Resultados ===\n"
            a_d_ratio = a / d
            results_output += f"1. Verificación 0.5 => a/d (a/d =
{a_d_ratio:.2f}): "
            if 0.5 < a_d_ratio:
                results_output += "Viola la condición de utilización del
método\n"
            else:
                results_output += "OK\n"

            # 2. Asv = (0.8*V)/(fy*μ)
            Asv = (0.8 * V) / (fy * mu)
            results_output += f"2. Armadura resistente a carga vertical
(Asv) = (0.8*V)/(fy*μ) = {Asv:.4f} mm²\n"

            # 3. As_t = Asv + (H/fy)
            As_t = Asv + (H / fy)
```

```
            results_output += f"3. Armadura del tirante (As_t) = Asv +
(H/fy) = {As_t:.4f} mm²\n"

            # 4. As_c = 0.5*(Asv/d)
            As_c = 0.5 * (Asv / d)
            results_output += f"4. Armadura de costura (As_c) =
0.5*(Asv/d) = {As_c:.4f} mm²\n"

            # 5. ρ = As_t/(b*d)
            rho = As_t / (b * d)
            results_output += f"5. Cuantía de armadura (ρ) = As_t/(b*d) =
{rho:.6f}\n"

            # 6. τwu = 3 + 0.9*ρ*fy
            tau_wu = 3 + 0.9 * rho * fy
            results_output += f"6. Tensión cizallante máxima (τwu) =  3 +
0.9*ρ*fy =  {tau_wu:.2f} MPa\n"

            # 7. tw1 = 0.27*((1-(fck/250))*fc
            tw1 = 0.27 * (1 - (fck / 250)) * fc
            results_output += f"7. Límite para tensión cizallante 1 (tw1) =
0.27*((1-(fck/250))*fc = {tw1:.2f} MPa\n"

            # 8. tw2 = 8 MPa
            tw2 = 8.0
            results_output += f"8. Límite para tensión cizallante 2 (tw2) =
{tw2:.2f} MPa\n"

            # 9. Determinar τwmax según las condiciones especificadas
            if tau_wu <= tw1 and tau_wu <= tw2:
                tau_wmax = tau_wu
                results_output += f"  τwmax = τwu = {tau_wmax:.2f} MPa
(porque τwu ≤ tw1 y τwu ≤ tw2)\n"
            else:
                tau_wmax = min(tw1, tw2)
                results_output += f"  τwmax = min(tw1, tw2) =
{tau_wmax:.2f} MPa (porque τwu > tw1 o τwu > tw2)\n"

            # 10. Verificación τw <= τwmax
            results_output += f"\n=== Verificación de la biela de
compresión ===\n"
            results_output += f"τw = {tau_w:.2f} MPa\n"
            results_output += f"τwmax = {tau_wmax:.2f} MPa\n"

            if tau_w <= tau_wmax:
                results_output += f"Verificación τw <= τwmax: OK - La biela
resiste ({tau_w:.2f} MPa ≤ {tau_wmax:.2f} MPa)\n"
            else:
                results_output += f"Verificación τw <= τwmax: NO CUMPLE -
La biela no resiste ({tau_w:.2f} MPa > {tau_wmax:.2f} MPa)\n"

            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, results_output)

        except ValueError:
            messagebox.showerror("Error de Entrada", "Por favor, ingrese
valores numéricos válidos en todos los campos.")
```

```
        except Exception as e:
            messagebox.showerror("Error de Cálculo", f"Ocurrió un error
durante el cálculo: {e}")

class ElDebbsConsolosCurtosGUI:
    def __init__(self, root, project_info=None):
        self.root = root
        self.root.title("El Debbs - Consolos Curtos")
        self.project_info = project_info if project_info else {}
        self.root.geometry("800x700")

        # Variables para los parámetros de entrada
        self.V = tk.DoubleVar()
        self.H = tk.DoubleVar()
        self.a = tk.DoubleVar()
        self.d = tk.DoubleVar()
        self.b = tk.DoubleVar()
        self.fy = tk.DoubleVar(value=500)
        self.fc = tk.DoubleVar(value=25)
        self.beta = tk.DoubleVar(value=1.0)

        self.create_widgets()

    def create_widgets(self):
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Frame de entrada de datos con scroll
        canvas = tk.Canvas(main_frame)
        scrollbar = ttk.Scrollbar(main_frame, orient="vertical",
command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        # Parámetros de entrada
        input_frame = ttk.LabelFrame(scrollable_frame, text="Factores de
seguridad", padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Modo de aplicación de carga
(β):").grid(row=0, column=0, sticky=tk.W, padx=5, pady=2)
        beta_frame = ttk.Frame(input_frame)
        beta_frame.grid(row=0, column=1, sticky=tk.W, padx=5, pady=2)
        ttk.Radiobutton(beta_frame, text="Directa (1.0)",
variable=self.beta, value=1.0).pack(side=tk.LEFT)
        ttk.Radiobutton(beta_frame, text="Indirecta (0.85)",
variable=self.beta, value=0.85).pack(side=tk.LEFT)

        input_frame = ttk.LabelFrame(scrollable_frame, text="Cargas
Actuantes", padding="10")
```

```
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Carga vertical (V) [N]:").grid(row=1,
column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.V, width=15).grid(row=1,
column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Carga Horizontal (H)
[N]:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.H, width=15).grid(row=2,
column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(scrollable_frame, text="Parámetros
Geométricos", padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Punto de aplicación de carga (a)
[mm]:").grid(row=3, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.a, width=15).grid(row=3,
column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Altura útil (d) [mm]:").grid(row=4,
column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.d, width=15).grid(row=4,
column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Ancho de la pieza (b)
[mm]:").grid(row=5, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.b, width=15).grid(row=5,
column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(scrollable_frame, text="Materiales",
padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Resistencia a fluencia del acero (fy)
[MPa]:").grid(row=6, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.fy, width=15).grid(row=6,
column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Resistencia a compresión del concreto
(fc) [MPa]:").grid(row=7, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.fc, width=15).grid(row=7,
column=1, padx=5, pady=2)

        # Botón de cálculo
        calc_button = ttk.Button(scrollable_frame, text="CALCULAR",
command=self.calculate, style="Accent.TButton")
        calc_button.pack(pady=10)

        # Frame de resultados unificado (como en NBR)
        results_frame = ttk.LabelFrame(scrollable_frame, text="Resultados",
padding="10")
        results_frame.pack(fill=tk.BOTH, expand=True, pady=5)

        self.results_text = scrolledtext.ScrolledText(results_frame,
height=40, width=80)
```

```
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Empacar canvas y scrollbar
        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

    def calculate(self):
        try:
            # Obtener valores de entrada
            V = self.V.get()
            H = self.H.get()
            a = self.a.get()
            d = self.d.get()
            b = self.b.get()
            fy = self.fy.get()
            fc = self.fc.get()
            beta = self.beta.get()

            # Iniciar construcción del reporte de resultados
            results_output = ""
            results_output += f"Fecha: {self.project_info.get('date',
'N/A')}\n"
            results_output += f"Nombre del Proyecto:
{self.project_info.get('project_name', 'N/A')}\n"
            results_output += f"Nombre del Usuario:
{self.project_info.get('user_name', 'N/A')}\n\n"

            results_output += "=== PARÁMETROS DE ENTRADA ===\n"
            results_output += f"V = {V} N\n"
            results_output += f"H = {H} N\n"
            results_output += f"a = {a} mm\n"
            results_output += f"d = {d} mm\n"
            results_output += f"b = {b} mm\n"
            results_output += f"fy = {fy} MPa\n"
            results_output += f"fc = {fc} MPa\n"
            results_output += f"β = {beta}\n\n"

            # 1. Verificar relación a/d
            a_d_ratio = a / d
            results_output += f"1. Relación a/d = {a_d_ratio:.3f}\n"
            if not (0.5 < a_d_ratio <= 1):
                results_output += "   ADVERTENCIA: La relación a/d no está
en el rango recomendado (0.5 < a/d ≤ 1)\n"

            # 2. Calcular armadura del tirante (As)
            As = ((V * a) / (0.9 * fy * d)) + 1.2 * (H / fy)
            results_output += f"2. Armadura del tirante (As) =
((V*a)/(0.9*fy*d)) + 1.2*(H/fy) = {As:.2f} mm²\n"

            # 3. Calcular armadura de costura (Asc)
            Asc = 0.5 * (1 / fy) * (V * a / (0.9 * d))
            results_output += f"3. Armadura de costura (Asc) =
0.5*(1/fy)*(V*a/(0.9*d)) = {Asc:.2f} mm²\n"

            # 4. Calcular tensión actuante (τwd)
            tau_wd = V / (b * d)
```

```
            results_output += f"4. Tensión actuante (τwd) = V/(b*d) =
{tau_wd:.4f} MPa\n"

            # 5. Calcular tensión límite (τwu)
            tau_wu = (0.18 * beta * fc) / math.sqrt(0.9**2 + a_d_ratio**2)
            results_output += f"5. Tensión límite (τwu) =
(0.18*β*fc)/√(0.9² + (a/d)²) = {tau_wu:.4f} MPa\n"

            # 6. Verificación
            results_output += "\n=== VERIFICACIÓN DE RESISTENCIA ===\n"
            if tau_wd <= tau_wu:
                results_output += f"τwd ({tau_wd:.4f} MPa) ≤ τwu
({tau_wu:.4f} MPa) → CUMPLE\n"
            else:
                results_output += f"τwd ({tau_wd:.4f} MPa) > τwu
({tau_wu:.4f} MPa) → NO CUMPLE\n"

            # Mostrar resultados en el cuadro de texto
            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, results_output)

        except ValueError:
            messagebox.showerror("Error", "Por favor ingrese valores
numéricos válidos en todos los campos.")
        except ZeroDivisionError:
            messagebox.showerror("Error", "No se puede dividir por cero.
Verifique los valores ingresados.")
        except Exception as e:
            messagebox.showerror("Error", f"Ocurrió un error durante el
cálculo: {str(e)}")

class ElDebbsMensulasMuyCortasGUI:
    def __init__(self, root, project_info=None):
        self.root = root
        self.root.title("El Debbs - Ménsulas muy Cortas")
        self.project_info = project_info if project_info else {}
        self.root.geometry("800x700")

        # Variables para los parámetros de entrada
        self.V = tk.DoubleVar()
        self.H = tk.DoubleVar()
        self.a = tk.DoubleVar()
        self.d = tk.DoubleVar()
        self.b = tk.DoubleVar()
        self.fy = tk.DoubleVar(value=500)
        self.fc = tk.DoubleVar(value=25)
        self.fck = tk.DoubleVar(value=30)
        self.mu = tk.DoubleVar(value=1.4)

        self.create_widgets()

    def create_widgets(self):
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Frame de entrada de datos con scroll
        canvas = tk.Canvas(main_frame)
```

```
        scrollbar = ttk.Scrollbar(main_frame, orient="vertical",
command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all"))
        )

        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        # Parámetros de entrada
        input_frame = ttk.LabelFrame(scrollable_frame, text="Factores de
seguridad", padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Coeficiente de rugosidad
(μ):").grid(row=0, column=0, sticky=tk.W, padx=5, pady=2)
        mu_frame = ttk.Frame(input_frame)
        mu_frame.grid(row=0, column=1, sticky=tk.W, padx=5, pady=2)
        ttk.Radiobutton(mu_frame, text="1.4 (Hormigón monolítico)",
variable=self.mu, value=1.4).pack(side=tk.LEFT)
        ttk.Radiobutton(mu_frame, text="1.0 (Hormigón sobre hormigón
endurecido)", variable=self.mu, value=1.0).pack(side=tk.LEFT)
        ttk.Radiobutton(mu_frame, text="0.6 (Interface lisa)",
variable=self.mu, value=0.6).pack(side=tk.LEFT)

        input_frame = ttk.LabelFrame(scrollable_frame, text="Cargas",
padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Carga vertical (V) [N]:").grid(row=1,
column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.V, width=15).grid(row=1,
column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Carga Horizontal (H)
[N]:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.H, width=15).grid(row=2,
column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(scrollable_frame, text="Parámetros
Geométricos", padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Punto de aplicación de carga (a)
[mm]:").grid(row=4, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.a, width=15).grid(row=4,
column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Altura útil (d) [mm]:").grid(row=5,
column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.d, width=15).grid(row=5,
column=1, padx=5, pady=2)
```

```
        ttk.Label(input_frame, text="Ancho de la pieza (b) [mm]:").grid(row=6, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.b, width=15).grid(row=6, column=1, padx=5, pady=2)

        input_frame = ttk.LabelFrame(scrollable_frame, text="Materiales", padding="10")
        input_frame.pack(fill=tk.X, pady=5)

        ttk.Label(input_frame, text="Resistencia a fluencia del acero (fy) [MPa]:").grid(row=7, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.fy, width=15).grid(row=7, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Resistencia a compresión del concreto (fc) [MPa]:").grid(row=8, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.fc, width=15).grid(row=8, column=1, padx=5, pady=2)

        ttk.Label(input_frame, text="Resistencia característica del concreto (fck) [MPa]:").grid(row=9, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(input_frame, textvariable=self.fck, width=15).grid(row=9, column=1, padx=5, pady=2)

        # Botón de cálculo
        calc_button = ttk.Button(scrollable_frame, text="CALCULAR", command=self.calculate, style="Accent.TButton")
        calc_button.pack(pady=10)

        # Frame de resultados (similar a NBR)
        results_frame = ttk.LabelFrame(scrollable_frame, text="Resultados", padding="10")
        results_frame.pack(fill=tk.BOTH, expand=True, pady=5)

        self.results_text = scrolledtext.ScrolledText(results_frame, height=30, width=80)
        self.results_text.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Empacar canvas y scrollbar
        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

    def calculate(self):
        try:
            # Obtener valores de entrada
            V = self.V.get()
            H = self.H.get()
            a = self.a.get()
            d = self.d.get()
            b = self.b.get()
            fy = self.fy.get()
            fc = self.fc.get()
            fck = self.fck.get()
            mu = self.mu.get()

            # Iniciar construcción del reporte de resultados
            results_output = ""
```

```
            results_output += f"Fecha: {self.project_info.get('date',
'N/A')}\n"
            results_output += f"Nombre del Proyecto:
{self.project_info.get('project_name', 'N/A')}\n"
            results_output += f"Nombre del Usuario:
{self.project_info.get('user_name', 'N/A')}\n\n"

            results_output += "=== PARÁMETROS DE ENTRADA ===\n"
            results_output += f"V = {V} N\n"
            results_output += f"H = {H} N\n"
            results_output += f"a = {a} mm\n"
            results_output += f"d = {d} mm\n"
            results_output += f"b = {b} mm\n"
            results_output += f"fy = {fy} MPa\n"
            results_output += f"fc = {fc} MPa\n"
            results_output += f"fck = {fck} MPa\n"
            results_output += f"μ = {mu}\n\n"

            # Verificar relación a/d
            a_d = a / d
            results_output += f"1. Relación a/d = {a_d:.3f}\n"
            if a_d >= 0.5:
                results_output += "   ADVERTENCIA: La relación a/d no
cumple con a/d < 0.5 para ménsulas muy cortas\n\n"

            # Calcular armadura del tirante (As)
            As = (1 / fy) * ((0.8 * V / mu) + H)
            results_output += f"2. Armadura del tirante (As) =
(1/fy)*((0.8*V/μ) + H) = {As:.2f} mm²\n"

            # Calcular armadura de costura (Asc)
            if d == 0:
                raise ZeroDivisionError("La altura útil (d) no puede ser
cero")
            Asc = 0.5 * (1 / fy) * (V * a / (0.9 * d))
            results_output += f"3. Armadura de costura (Asc) =
0.5*(1/fy)*(V*a/(0.9*d)) = {Asc:.2f} mm²\n"

            # Calcular cuantía (ρ)
            rho = As / (b * d)
            results_output += f"4. Cuantía (ρ) = As/(b*d) = {rho:.6f}\n"

            # Calcular τwd
            tau_wd = V / (b * d)
            results_output += f"5. Tensión actuante (τwd) = V/(b*d) =
{tau_wd:.4f} MPa\n"

            # Calcular τwu, tw1, tw2
            tau_wu = 3 + 0.9 * rho * fy
            results_output += f"6. Tensión límite (τwu) = 3 + 0.9*ρ*fy =
{tau_wu:.2f} MPa\n"

            tw1 = 0.27 * (1 - (fck / 250)) * fc
            results_output += f"7. tw1 = 0.27*(1-(fck/250))*fc = {tw1:.2f}
MPa\n"

            tw2 = 8.0  # Valor constante según especificación
```

```
            results_output += f"8. tw2 = 8.0 MPa (valor constante)\n"

            # Determinar τwmax
            if tau_wu <= tw1 and tau_wu <= tw2:
                tau_wmax = tau_wu
                results_output += f"9. Como τwu ≤ tw1 y τwu ≤ tw2 → τwmax =
τwu = {tau_wmax:.2f} MPa\n"
            else:
                tau_wmax = min(tw1, tw2)
                results_output += f"9. Como τwu > tw1 o τwu > tw2 → τwmax =
min(tw1,tw2) = {tau_wmax:.2f} MPa\n"

            # Verificación τwd <= τwmax
            results_output += "\n=== VERIFICACIÓN DE LA BIELA DE COMPRESIÓN
===\n"
            results_output += f"τwd = {tau_wd:.4f} MPa\n"
            results_output += f"τwmax = {tau_wmax:.2f} MPa\n"

            if tau_wd <= tau_wmax:
                results_output += f"Verificación: τwd ({tau_wd:.4f} MPa) ≤
τwmax ({tau_wmax:.2f} MPa) → CUMPLE\n"
            else:
                results_output += f"Verificación: τwd ({tau_wd:.4f} MPa) >
τwmax ({tau_wmax:.2f} MPa) → NO CUMPLE\n"

            # Mostrar resultados en el cuadro de texto
            self.results_text.delete(1.0, tk.END)
            self.results_text.insert(tk.END, results_output)

        except ValueError:
            messagebox.showerror("Error", "Por favor ingrese valores
numéricos válidos en todos los campos.")
        except ZeroDivisionError:
            messagebox.showerror("Error", "No se puede dividir por cero.
Verifique los valores ingresados.")
        except Exception as e:
            messagebox.showerror("Error", f"Ocurrió un error durante el
cálculo: {str(e)}")

class MainApplication(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Aplicación de Dimensionamiento Estructural")
        self.geometry("700x650")  # Aumenté el ancho y alto para las
imágenes

        self.project_date = tk.StringVar(value=date.today().strftime("%Y-
%m-%d"))
        self.project_name = tk.StringVar()
        self.user_name = tk.StringVar()

        # INICIALIZAR LA LISTA PRIMERO, antes de cualquier otro método
        self.image_references = []

        self.create_widgets()

    def create_widgets(self):
```

```
        main_frame = ttk.Frame(self, padding="10")
        main_frame.pack(fill=tk.BOTH, expand=True)

        # Sección de Datos Personales
        personal_data_frame = ttk.LabelFrame(main_frame, text="Datos del
Proyecto", padding="10")
        personal_data_frame.pack(fill=tk.X, pady=10)

        ttk.Label(personal_data_frame, text="Fecha:").grid(row=0, column=0,
sticky=tk.W, padx=5, pady=2)
        ttk.Entry(personal_data_frame, textvariable=self.project_date,
width=30).grid(row=0, column=1, padx=5, pady=2)

        ttk.Label(personal_data_frame, text="Nombre del
Proyecto:").grid(row=1, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(personal_data_frame, textvariable=self.project_name,
width=30).grid(row=1, column=1, padx=5, pady=2)

        ttk.Label(personal_data_frame, text="Nombre del
Usuario:").grid(row=2, column=0, sticky=tk.W, padx=5, pady=2)
        ttk.Entry(personal_data_frame, textvariable=self.user_name,
width=30).grid(row=2, column=1, padx=5, pady=2)

        # Sección de Selección de Método
        method_selection_frame = ttk.LabelFrame(main_frame,
text="Seleccionar Método de Cálculo", padding="10")
        method_selection_frame.pack(fill=tk.X, pady=5)

        ttk.Button(method_selection_frame, text="PCI - Método STM",
command=self.open_pci_stm_method, style="Accent.TButton").pack(pady=5)
        ttk.Button(method_selection_frame, text="PCI - Viga en Balance",
command=self.open_pci_cantilever_method,
style="Accent.TButton").pack(pady=5)
        ttk.Button(method_selection_frame, text="NBR - Ménsulas Cortas",
command=self.open_nbr_consolos_curtos_method,
style="Accent.TButton").pack(pady=5)
        ttk.Button(method_selection_frame, text="NBR - Ménsulas muy
Cortas", command=self.open_nbr_consolos_muito_curtos_method,
style="Accent.TButton").pack(pady=5)
        ttk.Button(method_selection_frame, text="El Debbs - Ménsulas
Cortas", command=self.open_el_debbs_consolos_curtos_method,
style="Accent.TButton").pack(pady=5)
        ttk.Button(method_selection_frame, text="El Debbs - Ménsulas muy
Cortas", command=self.open_el_debbs_mensulas_muy_cortas_method,
style="Accent.TButton").pack(pady=5)

        # Sección de Imágenes
        images_frame = ttk.LabelFrame(main_frame, text="Diagramas
Estructurales", padding="10")
        images_frame.pack(fill=tk.BOTH, expand=True, pady=10)

        # Frame para contener las dos imágenes lado a lado
        images_container = ttk.Frame(images_frame)
        images_container.pack(fill=tk.BOTH, expand=True, pady=10)

        # Primera imagen (lado izquierdo)
        image1_frame = ttk.Frame(images_container)
```

```
        image1_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=(0,
5))

        ttk.Label(image1_frame, text="Modelo 3D", font=('Arial', 10,
'bold')).pack(pady=(0, 5))

        try:
            # VERIFICA que esta ruta sea correcta
            imagen1_original = Image.open("++++++++++++++")
            imagen1_original = imagen1_original.resize((300, 300),
Image.Resampling.LANCZOS)
            img1_ref = ImageTk.PhotoImage(imagen1_original)
            self.image_references.append(img1_ref)  # Guardar referencia

            image1_label = ttk.Label(image1_frame, image=img1_ref)
            image1_label.pack(pady=5)

        except Exception as e:
            print(f"Error cargando imagen 1: {e}")  # Debug en consola
            ttk.Label(image1_frame, text=f"Error al cargar imagen 1: {e}",
foreground="red").pack(pady=5)

        # Segunda imagen (lado derecho)
        image2_frame = ttk.Frame(images_container)
        image2_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True,
padx=(5, 0))

        ttk.Label(image2_frame, text="Esquema de armaduras", font=('Arial',
10, 'bold')).pack(pady=(0, 5))

        try:
            # VERIFICA que esta ruta sea correcta - CAMBIA por tu segunda
imagen real
            imagen2_original = Image.open("++++++++++++++")
            imagen2_original = imagen2_original.resize((300, 300),
Image.Resampling.LANCZOS)
            img2_ref = ImageTk.PhotoImage(imagen2_original)
            self.image_references.append(img2_ref)  # Guardar referencia

            image2_label = ttk.Label(image2_frame, image=img2_ref)
            image2_label.pack(pady=5)

        except Exception as e:
            print(f"Error cargando imagen 2: {e}")  # Debug en consola
            # Si no tienes segunda imagen, usa un placeholder
            ttk.Label(image2_frame, text="Diagrama de cálculo no
disponible", foreground="blue").pack(pady=5)

    def open_pci_stm_method(self):
        project_info = {
            "date": self.project_date.get(),
            "project_name": self.project_name.get(),
            "user_name": self.user_name.get()
        }
        new_window = tk.Toplevel(self)
        TrussCalculatorGUI(new_window, project_info)
```

```
    def open_pci_cantilever_method(self):
        project_info = {
            "date": self.project_date.get(),
            "project_name": self.project_name.get(),
            "user_name": self.user_name.get()
        }
        new_window = tk.Toplevel(self)
        PCICantileverGUI(new_window, project_info)

    def open_nbr_consolos_curtos_method(self):
        project_info = {
            "date": self.project_date.get(),
            "project_name": self.project_name.get(),
            "user_name": self.user_name.get()
        }
        new_window = tk.Toplevel(self)
        NBRConsolosCurtosGUI(new_window, project_info)

    def open_nbr_consolos_muito_curtos_method(self):
        project_info = {
            "date": self.project_date.get(),
            "project_name": self.project_name.get(),
            "user_name": self.user_name.get()
        }
        new_window = tk.Toplevel(self)
        NBRConsolosMuitoCurtosGUI(new_window, project_info)

    def open_el_debbs_consolos_curtos_method(self):
        project_info = {
            "date": self.project_date.get(),
            "project_name": self.project_name.get(),
            "user_name": self.user_name.get()
        }
        new_window = tk.Toplevel(self)
        ElDebbsConsolosCurtosGUI(new_window, project_info)

    def open_el_debbs_mensulas_muy_cortas_method(self):
        project_info = {
            "date": self.project_date.get(),
            "project_name": self.project_name.get(),
            "user_name": self.user_name.get()
        }
        new_window = tk.Toplevel(self)
        ElDebbsMensulasMuyCortasGUI(new_window, project_info)

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