SISTEMA COMPUTACIONAL PARA SIMULAÇÃO 2D DE VIGAS BIAPOIADAS COM CARGAS PONTUAIS E DISTRIBUÍDAS

João Pedro dos Santos Martins, João Victor de Campos, Hélio Lourenço Esperidião Ferreira.

Universidade do Vale do Paraíba/Curso de Engenharia da Computação, Avenida Shishima Hifumi, 2911, Urbanova - 12244-000 - São José dos Campos-SP, Brasil, johnmartines.sm@gmail.com.

Resumo

Este trabalho apresenta o desenvolvimento de um simulador computacional em Python para análise didática de vigas 2D biapoiadas. A aplicação permite definir o comprimento da viga, inserir cargas pontuais e distribuídas e visualizar apoios, carregamentos e reações. A interface foi construída com Tkinter e organizada em módulos de visualização, controle, modelo, cálculo, renderização e persistência, com separação próxima ao padrão Model-View-Controller e uso de Data Access Object para o banco local. As reações são obtidas pelas equações de equilíbrio estático, considerando pino em A e rolete em B, e os projetos são armazenados em SQLite com dados serializados em JSON. A rotina de cálculo foi verificada em cinco casos com solução analítica conhecida, e os resultados coincidiram dentro da precisão numérica esperada. O simulador funciona como ferramenta didática de código aberto, enquanto os diagramas de esforço cortante e momento fletor permanecem como continuidade do projeto.

Palavras-chave: Vigas biapoiadas. Estática. Python. Simulação. Interface gráfica.

Área do Conhecimento: Engenharias IV - Engenharia Elétrica e Engenharia da Computação.

Introdução

A análise de vigas é um conteúdo recorrente em disciplinas de Estática e Resistência dos Materiais, pois reúne conceitos de força, momento, equilíbrio e carregamento distribuído. Antes do cálculo de tensões ou deformações, é necessário determinar corretamente as reações nos apoios a partir das condições de equilíbrio do corpo rígido. Em uma viga plana biapoiada, essas condições são normalmente expressas pelas somas de forças nas direções horizontal e vertical e pela soma dos momentos, que devem ser iguais a zero (Hibbeler, 2022).

Uma dificuldade comum nas primeiras disciplinas de estruturas é transformar o enunciado e as equações em uma representação física coerente. Para quem ainda está aprendendo, entender onde cada força atua e como ela interfere nas reações pode ser tão importante quanto resolver as contas. Johnsen et al. (2016), ao estudar uma ferramenta de visualização para teoria clássica de vigas, apontam que recursos visuais podem contribuir para a compreensão conceitual de alunos de engenharia. Essa foi uma das motivações do projeto: fazer com que o estudante veja no mesmo ambiente a viga, os apoios, as cargas inseridas e as reações calculadas.

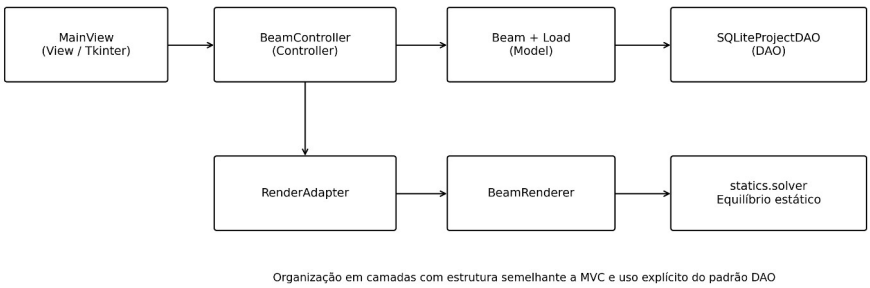
Existem ferramentas muito mais completas para análise estrutural, incluindo soluções comerciais voltadas ao uso profissional. Elas são importantes principalmente quando o problema envolve dimensionamento, normas, elementos finitos e integração com outros softwares. Para o primeiro contato com equilíbrio de vigas, porém, uma ferramenta menor pode ter outra função: deixar evidente quais dados foram inseridos e como eles entram no cálculo, sem exigir que o aluno domine, ao mesmo tempo, um ambiente profissional inteiro.

Dessa forma, o objetivo deste trabalho foi desenvolver e avaliar uma aplicação gráfica para vigas biapoiadas que permita inserir diferentes carregamentos, calcular as reações de equilíbrio, visualizar o problema e salvar projetos para uso posterior. Além do resultado numérico, buscou-se manter uma estrutura de código simples o suficiente para ser acompanhada em contexto acadêmico e disponibilizada publicamente para consulta e continuidade do desenvolvimento.

Metodologia

O software foi desenvolvido em Python e dividido em módulos de interface, controle, modelo, cálculo, renderização e persistência. A organização adotada segue uma separação de responsabilidades próxima ao padrão arquitetural Model-View-Controller (MVC), no qual apresentação, controle e modelo são mantidos em componentes distintos (Buschmann et al., 1996). No simulador, a MainView concentra os elementos da interface, o BeamController recebe as ações do usuário e coordena o fluxo, e as classes Beam e Load representam os dados do problema. O acesso ao banco foi isolado na classe SQLiteProjectDAO, seguindo o padrão Data Access Object (DAO), empregado para encapsular a comunicação com a fonte de dados (Alur; Crupi; Malks, 2003). O solver de equilíbrio e o renderizador permanecem fora da interface, reduzindo o acoplamento entre apresentação, cálculo e persistência.

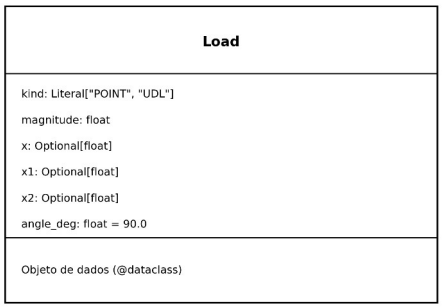
Figura 1 - Organização dos principais módulos e padrões utilizados no simulador.



Fonte: elaborado pelos autores a partir do código do projeto (2026).

A classe Load foi implementada como dataclass e reúne os dados necessários para os dois carregamentos aceitos pela aplicação. Em uma carga pontual, são utilizados magnitude, posição e ângulo. Na carga distribuída uniforme, o objeto armazena a intensidade por unidade de comprimento e os limites inicial e final. A Figura 2 mostra essa estrutura de forma isolada. Essa escolha evita manter duas estruturas independentes de carga e simplifica o tratamento realizado pelo controlador.

Figura 2 - Diagrama simplificado da classe Load.



Fonte: elaborado pelos autores a partir de models/load.py (2026).

Para uma carga pontual de magnitude F, a rotina de cálculo decompõe a força em Fx = F cos(θ) e Fy = -F sen(θ), adotando sinal negativo para forças verticais direcionadas para baixo. Para uma carga distribuída uniforme de intensidade w entre x1 e x2, o programa utiliza a força equivalente Feq = w(x2 - x1), aplicada no centro do intervalo, xeq = (x1 + x2)/2. Como o apoio B é modelado como rolete, Bx = 0. A reação horizontal no pino é calculada por Ax = -ΣFx. Pelo equilíbrio de momentos em torno de A, obtém-se By = -ΣMA/L e, em seguida, Ay = -ΣFy - By.

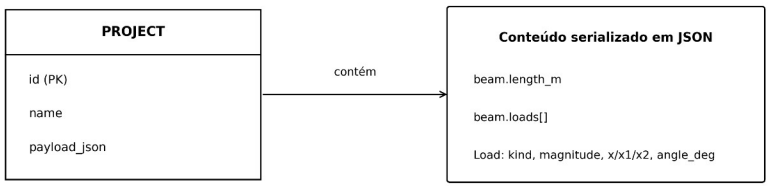
Tabela 1 - Principais módulos e responsabilidades do software.

Módulo	Tecnologia/arquivo	Responsabilidade
Interface	Tkinter / main_view.py	Entrada dos dados e comandos do usuário.
Controle	beam_controller.py	Coordenação das operações e atualização do modelo.
Modelo	beam.py / load.py	Representação da viga e dos carregamentos.
Cálculo	statics/solver.py	Decomposição de forças e reações de apoio.
Renderização	renderer.py	Desenho da viga, cargas, apoios e reações.
Persistência	SQLite / sqlite_dao.py	Salvar e recuperar projetos em JSON.

Fonte: elaborado pelos autores (2026).

A persistência usada pelo fluxo principal do programa é feita com sqlite3 por meio da classe SQLiteProjectDAO. Cada projeto é registrado na tabela project com id, nome e um campo payload_json. Nesse campo são serializados o comprimento da viga e a lista de carregamentos em JSON, formato textual padronizado para intercâmbio de dados (Bray, 2017). Para a versão atual, essa solução é simples e suficiente, pois permite salvar e reconstruir o estado do projeto sem distribuir os dados em muitas tabelas. A Figura 3 resume esse modelo.

Figura 3 - Modelo de persistência utilizado pela classe SQLiteProjectDAO.



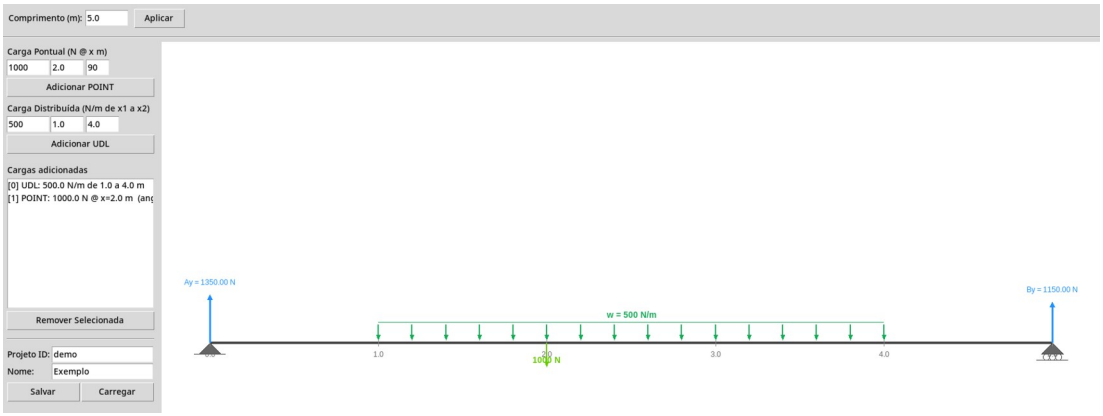
Fonte: elaborado pelos autores a partir de dao/sqlite_dao.py (2026).

A verificação da rotina estática foi realizada com cinco casos de comprimento L = 5 m: carga pontual central de 1000 N; carga pontual de 1000 N em x = 2 m; carga distribuída de 500 N/m entre 1 e 4 m; combinação das duas últimas cargas; e força inclinada de 1000 N a 60° em x = 2 m. Os resultados retornados pelo programa foram comparados diretamente com os valores obtidos pelas equações de equilíbrio. Também foi salvo e recuperado um projeto de teste para verificar a persistência.

Resultados

Durante os testes, a interface permitiu alterar o comprimento da viga e incluir cargas dos tipos POINT e UDL sem reiniciar o programa. A cada mudança, o canvas foi redesenhado com os apoios, as cargas e as reações correspondentes. A Figura 4 mostra um exemplo com uma carga distribuída de 500 N/m entre 1 e 4 m e uma carga pontual vertical de 1000 N em x = 2 m. Nessa configuração, foram obtidos Ay = 1350 N e By = 1150 N, com reação horizontal numericamente próxima de zero.

Figura 4 - Interface do simulador com carga pontual, carga distribuída e reações calculadas.



Fonte: captura da aplicação desenvolvida pelos autores (2026).

Os cinco casos de verificação apresentaram concordância com os resultados analíticos. Para cargas verticais a 90°, o valor de Ax retornado pelo programa pode aparecer da ordem de 10^{-14} N devido à representação de ponto flutuante de cos(90°), sendo tratado como zero na interpretação dos resultados. No caso inclinado, a componente horizontal foi equilibrada pela reação Ax do apoio de pino.

Tabela 2 - Verificação das reações calculadas pelo simulador para L = 5 m.

Caso	Ax (N)	Ay (N)	By (N)	Resultado
1000 N em x=2,5 m	0,00	500,00	500,00	Conforme
1000 N em x=2,0 m	0,00	600,00	400,00	Conforme
500 N/m em [1,4]	0,00	750,00	750,00	Conforme
POINT + UDL	0,00	1350,00	1150,00	Conforme
1000 N, 60°, x=2 m	-500,00	519,62	346,41	Conforme

Fonte: testes realizados no código do projeto (2026).

A persistência também foi verificada. Um projeto com comprimento de 7,5 m, carga pontual de 800 N em x = 3 m e carga distribuída de 200 N/m entre 1 e 5 m foi salvo e carregado por uma nova instância do controlador. Os valores recuperados corresponderam aos dados gravados, indicando que a serialização e a reconstrução dos objetos funcionam para os tipos de carga presentes na versão atual.

Discussão

O SkyCiv Beam oferece um conjunto mais amplo de resultados para vigas, incluindo reações, diagramas de esforço cortante e momento fletor, deflexões e tensões. A plataforma mantém uma calculadora de vigas gratuita e também possui planos comerciais com recursos adicionais. Para o estudante, isso é útil principalmente para conferir resultados e avançar para análises que o protótipo deste trabalho ainda não realiza. A proposta aqui é diferente: o simulador é executado localmente, não exige aquisição de licença do próprio programa e deixa o código-fonte publicamente disponível, permitindo que a rotina de equilíbrio também seja estudada como parte da atividade acadêmica (SkyCiv, 2026a; SkyCiv, 2026b).

O Autodesk Robot Structural Analysis Professional está em outro nível de aplicação. O software foi desenvolvido para análise e projeto estrutural profissional e inclui análises estáticas, modais e não lineares, malha de elementos finitos, verificações por normas e integração com o Revit. Trata-se de uma solução comercial, embora a Autodesk ofereça acesso educacional gratuito, por períodos renováveis, a estudantes e educadores elegíveis. Assim, o custo pode deixar de ser uma barreira em instituições qualificadas, mas o número de recursos, configurações e conceitos envolvidos continua sendo muito maior do que o necessário para uma primeira atividade de equilíbrio de vigas (Autodesk, 2026a; Autodesk, 2026b).

No uso acadêmico, portanto, os três programas não precisam ser vistos como concorrentes diretos. SkyCiv e Robot são alternativas mais adequadas quando a disciplina já exige diagramas, deformações, dimensionamento ou fluxos profissionais. O simulador desenvolvido é mais limitado, mas essa limitação pode ser vantajosa nas etapas iniciais: o aluno trabalha apenas com comprimento, apoios, carregamentos e equilíbrio, e o professor ainda pode usar o próprio código para mostrar como as equações foram implementadas. A Tabela 3 resume essa diferença de finalidade.

Tabela 3 - Comparação do simulador com ferramentas comerciais no contexto acadêmico.

Sistema	Acesso	Escopo principal	Uso acadêmico
Simulador desenvolvido	Execução local; sem aquisição de licença comercial; código público no GitHub	Viga biapoiada, cargas POINT/UDL e reações	Introdução ao equilíbrio e estudo do próprio código
SkyCiv Beam	Calculadora gratuita e planos comerciais	Reações, V(x), M(x), deflexão, tensão e recursos de projeto	Conferência e avanço para análises de viga mais completas
Autodesk Robot	Produto comercial; acesso educacional gratuito para elegíveis	Análise estrutural profissional, FEM, normas e BIM	Formação em ferramentas profissionais e problemas mais complexos

Fonte: elaborado pelos autores com base em SkyCiv (2026a, 2026b) e Autodesk (2026a, 2026b).

Dentro do escopo que foi implementado, os testes mostraram que as equações de equilíbrio estão sendo aplicadas de forma coerente. A separação entre interface, controlador, modelos, solver, renderizador e DAO também se mostrou útil durante o desenvolvimento: uma mudança visual não exige alterar a rotina de cálculo, e a persistência pode ser revista sem modificar a representação gráfica. Para um projeto acadêmico que ainda deverá receber novas funções, essa separação torna as próximas etapas mais controláveis.

As limitações atuais também ajudam a definir a próxima versão. O programa ainda calcula apenas as reações externas e não gera os diagramas de esforço cortante V(x) e momento fletor M(x). Além disso, as validações existentes em utils/validation.py precisam ser ligadas de forma consistente ao controlador para impedir posições de carga fora do comprimento da viga. Esses dois pontos são prioritários porque afetam diretamente o valor didático e a robustez da aplicação.

Outro ajuste necessário é a persistência. O repositório ainda mantém duas estruturas: dao/sqlite_dao.py, usada pelo controlador e baseada na tabela project com JSON, e dao/database.py, que cria as tabelas loads e settings. A intenção para a continuidade é manter uma única estratégia, acrescentar testes automatizados para as equações de equilíbrio e atualizar o README para refletir o que o programa realmente faz hoje.

Como evolução do projeto, a próxima etapa deverá implementar V(x) e M(x) e apresentar esses diagramas em uma área gráfica própria. Depois disso, podem ser acrescentados cálculo de deflexão, exemplos didáticos prontos, exportação dos resultados e uma opção de mostrar o desenvolvimento das equações passo a passo. Essa última função seria especialmente interessante para uso em sala, porque permitiria ao aluno comparar o cálculo manual com o resultado do programa sem transformar a ferramenta em uma “caixa-preta”. Em uma etapa posterior, uma avaliação com estudantes poderia investigar usabilidade e contribuição para a aprendizagem, observando os procedimentos éticos aplicáveis caso haja coleta estruturada com participantes.

Conclusão

Foi desenvolvido e avaliado um simulador 2D de vigas biapoiadas em Python capaz de representar cargas pontuais e distribuídas, calcular reações de apoio, atualizar graficamente a configuração estrutural e salvar projetos em banco SQLite. Os cinco casos de carregamento apresentaram os mesmos valores obtidos pelas equações de equilíbrio dentro da precisão numérica utilizada, e o teste de persistência recuperou corretamente o projeto salvo.

A arquitetura semelhante a MVC, associada ao uso de DAO, manteve as responsabilidades do software separadas e criou uma base adequada para continuidade. No contexto acadêmico, o principal diferencial não está em superar plataformas comerciais, mas em oferecer um caminho mais curto entre o problema de Estática, o resultado apresentado e o código que executa o cálculo. Como próximos passos, o projeto deverá incluir os diagramas V(x) e M(x), reforçar as validações de entrada, unificar a persistência e acrescentar recursos didáticos que mostrem o raciocínio de equilíbrio de forma mais explícita.

Referências

ALUR, D.; CRUPI, J.; MALKS, D. **Core J2EE Patterns: Best Practices and Design Strategies.** 2. ed. Upper Saddle River: Prentice Hall, 2003.

AUTODESK. **Robot Structural Analysis Professional: product details and key features.** 2026a. Disponível em: https://www.autodesk.com/products/robot-structural-analysis/product-details. Acesso em: 24 ago. 2026.

AUTODESK. **Autodesk Education: visão geral do plano educacional.** 2026b. Disponível em: https://www.autodesk.com/br/support/account/education/students-educators/overview. Acesso em: 24 ago. 2026.

BRAY, T. **The JavaScript Object Notation (JSON) Data Interchange Format.** RFC 8259. Internet Engineering Task Force, 2017. Disponível em: https://www.rfc-editor.org/rfc/rfc8259. Acesso em: 29 set. 2026.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. **Pattern-Oriented Software Architecture: A System of Patterns.** Chichester: Wiley, 1996.

HIBBELER, R. C. **Engineering Mechanics: Statics.** 15. ed. Pearson, 2022.

JOHNSEN, E.; NILSEN, M.; HJELSETH, E.; MERSCHBROCK, C. **Exploring a Simple Visualization Tool for Improving Conceptual Understanding of Classical Beam Theory.** Procedia Engineering, v. 164, p. 172-179, 2016. DOI: 10.1016/j.proeng.2016.11.607.

LITTLEOGINIRI. **Projeto-de-Vigas-fixas-2D: Simulação de Vigas 2D (Apoios Simples).** GitHub, 2026. Disponível em: https://github.com/LittleOginiri/Projeto-de-Vigas-fixas-2D. Acesso em: 24 ago. 2026.

PYTHON SOFTWARE FOUNDATION. **tkinter — Python interface to Tcl/Tk.** Python 3.13 documentation, 2026a. Disponível em: https://docs.python.org/3.13/library/tkinter.html. Acesso em: 24 ago. 2026.

PYTHON SOFTWARE FOUNDATION. **sqlite3 — DB-API 2.0 interface for SQLite databases.** Python 3.13 documentation, 2026b. Disponível em: https://docs.python.org/3.13/library/sqlite3.html. Acesso em: 24 ago. 2026.

SKYCIV. **Free Online Beam Calculator: reactions, shear force, bending moment, deflection and stress.** 2026a. Disponível em: https://skyciv.com/free-beam-calculator/. Acesso em: 24 ago. 2026.

SKYCIV. **Structural Design Software Professional Account.** 2026b. Disponível em: https://skyciv.com/pricing/professional-account/. Acesso em: 24 ago. 2026.