FUNDAÇÃO ALAGOANA DE PESQUISA, EDUCAÇÃO E CULTURA – FAPEC
Centro Universitário Mario Pontes Jucá – UMJ/FAT
Curso de Graduação em Engenharia Civil

Bruno Santos Lindoso
Maycon Júnior Teixeira da Silva

OTIMIZAÇÃO DE TRELIÇAS PLANAS COM PYTHON

MACEIÓ – AL
2020

BRUNO SANTOS LINDOSO
MAYCON JÚNIOR TEIXEIRA DA SILVA

OTIMIZAÇÃO DE TRELIÇAS PLANAS COM PYTHON

Trabalho de Conclusão de Curso (TCC) apresentado à Centro Universitário Mario Pontes Jucá – UMJ/FAT, como parte das exigências do Curso de Engenharia Civil – EC, para obtenção do título de Bacharelado.

Orientador:
Prof. Me. Luiz Carlos Lima Véras

MACEIÓ – AL
2020

BRUNO SANTOS LINDOSO
MAYCON JUNIOR TEIXEIRA DA SILVA

OTIMIZAÇÃO DE TRELIÇAS PLANAS COM PYTHON

Trabalho de Conclusão de Curso (TCC) apresentado à Centro Universitário Mario Pontes Jucá – UMJ/FAT, como parte da exigência para a conclusão do Curso de Engenharia Civil - EC, para obtenção do título de bacharel, aprovada em: 21 de dezembro de 2020.

Prof.Me. Luiz Carlos Lima Véras
Orientador – UMJ/FAT - AL

Banca Examinadora:

Prof.Me. Thiago Barbosa
Examinador Externo – UFAL - AL

Prof.Me. Jonas Cavalcante
Examinador Interno – UMJ/FAT - AL

MACEIÓ – AL
2020

Dedico este trabalho à Deus
e aos meus familiares e amigos.

AGRADECIMENTOS

Em primeiro lugar, a Deus, que fez com que nossos objetivos fossem alcançados, durante todos os nossos anos de estudos.

Aos nossos pais e irmãos e companheiras, que me incentivaram nos momentos difíceis e compreenderam nossa minha ausência enquanto nós nos dedicávamos à realização deste trabalho.

Ao professor Luiz Carlos Lima Véras, por ter sido nosso orientador e ter desempenhado tal função com dedicação e amizade, sempre disposto a ajudar e sanar as dúvidas.

À instituição de ensino, Centro Universitário Mario Pontes Jucá, essencial no nosso processo de formação profissional, pela dedicação, e por tudo o que aprendi ao longo dos anos do curso.

Eu, Bruno, agradeço aos meus avós Joaquim e Clébia, assim como meus pais, Alexandre e Suzanne, por todo o incentivo, desde o processo seletivo até a conclusão do curso. Se não fosse por eles, não seria possível realizar o sonho de me tronar Engenheiro Civil. À minha esposa Elaine e minha filha Lis por terem me apoiado (e aguentado minha ausência) durante os 5 anos que precisei me dedicar ao curso. Por fim, ao meu grande amigo Maycon, quem esteve comigo durante todo o curso, desde o primeiro até o último período, a quem recorri diversas vezes na hora das dúvidas.

Por fim eu Maycon, agradeço aos meus pais, João e Rosiene que foram fundamentais para minha carreira acadêmica, sempre apoiando meu sonho de ser engenheiro, meus irmãos Renata, Wilson e Danielle e minha noiva Natalia, que me incentivou nos momentos difíceis e compreendeu minha ausência enquanto me dedicava à realização deste trabalho. Ao meu amigo Bruno, por ter aceitado dividir comigo esse grande desafio aliando estruturas com tecnologia, aos meus amigos em especial ao Dênis, Lucas e Abmael, que durante o período do trabalho me deram incentivos e apoio para a conclusão do mesmo.

“Só se pode alcançar um grande êxito quando nos mantemos fiéis a nós mesmos.”
(Friedrich Nietzsche)

RESUMO

O superdimensionamento das estruturas não garante redução de material e desempenho estrutural ótimo e, podendo causar riscos e/ou perdas financeiras. Pesquisa-se sobre otimização estrutural aliada a métodos computacionais, a fim de otimizar a estrutura e obter um modelo que possa desempenhar a função de resistir aos esforços solicitados, com economia e qualidade chamado esse de “modelo ótimo”. Para isto é necessário um estudo detalhado dos métodos de rigidez direta, em conjunto com a linguagem computacional Python para que seja definido o tipo de otimização a ser executado. Realizou-se então um estudo de caso no qual verificou-se a otimização de uma treliça uniforme e seu desempenho com situações de mudança de carregamento, alteração da malha da treliça e retirada de nós e barras moldando diferentes tipos de treliças otimizadas conforme a carga exercida, o que impõe a constatação que o procedimento satisfaz o problema de otimização gerando treliças com a melhor configuração para o carregamento imposto.

Palavras-Chaves: treliça, análise estrutural, mecânica computacional.

ABSTRACT

The structure's oversizing does not guarantee material reduction and optimal structural performance and, it may cause risks and/or financial losses. Researchs of structural optimization combined with computational methods, in order to optimize the structure and obtain a model that can perform the function of resisting the requested efforts , with economy and quality called this "optimal model". For this, a detailed study of the direct stiffness methods is necessary, together with the Python computational language to define the type of optimization to be performed. Then, a case study was carried out in which the optimization of a uniform truss and its performance with changing load situations, alteration of the truss mesh and removal of nodes and bars shaping different types of trusses optimized according to the load performed , which imposes the observation that the procedure satisfies the optimization problem by generating trusses with the best configuration for the imposed load.

Keywords: truss, structural analysis, computational mechanics.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SÍMBOLOS

θ_x	Ângulo de inclinação do eixo x
θ_y	Ângulo de inclinação do eixo y
A	Área da seção transversal
ABNT	Associação Brasileira de Normas Técnicas
AG	Algoritmo Genérico
UMJ	Centro Universitário Mario Pontes Jucá
L	Comprimento da barra
λ_x	Cosseno de x
λ_y	Cosseno de y
d_F	Deslocamento em F
d_N	Deslocamento em N
$\mathbf{D}$	Deslocamento global
$\mathbf{d}$	Deslocamento local dos membros
$D_{Fx,y}$	Deslocamento no ponto F do eixo x ou y
$D_{Nx,y}$	Deslocamento no ponto N do eixo x ou y
FAT	Faculdade de Tecnologia de Alagoas
$F_{1,2}$	Força 1 ou 2
Q	Força global da estrutura
$\boldsymbol{q}$	Força global do membro
$Q_{Fx,y}$	Força global no ponto F do eixo x ou y
$Q_{Nx,y}$	Força global no ponto N do eixo x ou y
q_F	Força resultante em F
q_N	Força resultante em N
$\mathbf{K}$	Matriz de rigidez da estrutura
$\mathbf{k}$	Matriz de rigidez do membro em coordenadas globais
$\mathbf{k}'$	Matriz de rigidez dos membros
$\mathbf{T}$	Matriz de transformação
$\mathbf{T}^T$	Matriz transposta
E	Módulo de Elasticidade
$x' e\ y'$	Sistema de coordenadas locais

SUMÁRIO

1 INTRODUÇÃO

A otimização estrutural consiste na busca de melhores resultados, conseguindo o máximo ou mínimo de uma função matemática, com baixo custo e com melhor desempenho mecânico, já que os deslocamentos dos nós e as tensões atuantes nas barras são limitadores para a otimização (Silva, 2011). De acordo com (Pizzirani,2003), existem três tipos de abordagem para o problema de otimização estrutural: otimização topológica, otimização paramétrica e otimização de forma. A otimização topológica é usada quando não existe um perfil previamente estabelecido para a geometria, o processo para esta otimização é realizado com um modelo genérico. A otimização paramétrica usa uma ou mais variáveis de projeto assim parametrizando por meio de uma função objetivo a fim de minimizar e maximizar a mesma. Já a otimização de forma consiste em encontrar o domínio ótimo do problema, em que as coordenadas dos nós são as variáveis.

Avaliar a efetividade dos métodos computacionais na otimização de estruturas reticuladas por meio de AG (Algoritmo Genérico), que tem por base a probabilidade, tem alcançado muita popularidade nos dias de hoje, onde a tecnologia cresce e a demanda por resultados satisfatórios e com baixo custo são um diferencial.

O trabalho parte da hipótese que o superdimensionamento das estruturas não garante estabilidade global da estrutura, onde o aumento das seções e números de barras acarreta num maior peso na estrutura causando riscos à segurança e um oneroso custo.

Pretende-se tratar a otimização de estruturas reticuladas do gênero treliça. Nesse processo busca-se a melhor combinação probatória, que atenda aos esforços solicitantes com a mínima dimensão geométrica mantendo a melhor rigidez possível. Para isso, utilizou-se como ferramenta de programação a linguagem *Python* e as Bibliotecas *Openopt* e *NLopt.*

1.1 Objetivos

Este trabalho aplica linguagem computacional Python conjuntamente com AGs para as otimizações de topologia em estruturas reticuladas do gênero treliça. Os objetivos deste trabalho são:

Definir critérios que favoreçam o processo de otimização estrutural e uma metodologia de análise para o uso da linguagem computacional Python.

Otimizar a estrutura uniforme para que não exceda o limite de volume predefinido.

Para alcançar estes objetivos, foi necessário a realização das fases:

- Estudo do método de rigidez direta aplicado a treliça;
- Estudo da linguagem computacional Python;
- Estudo de otimização de topologia aplicada em treliças;
- Listar os métodos e técnicas a serem utilizados a fim de obter uma solução otimizada;
- Atestar os dados obtidos com a linguagem computacional com a literatura.

1.2 Estrutura do Trabalho

A monografia é composta de 7 capítulos. Para uma visão geral do trabalho a seguir, é apresentado uma prévia de cada capítulo.

O capítulo 1 apresenta a introdução, destacando o contexto do tema abordado. O capítulo 2 apresenta referência bibliográfica do método de otimização estrutural. O capítulo 3 mostra os tipos de otimizações existentes. O capitulo 4 se refere ao método de rigidez direta aplicado a treliças. O capítulo 5 apresenta técnicas de otimização de estruturas reticuladas. O capítulo 6 mostra as rotinas de computacionais para otimização de treliça, com linguagem Python. O capítulo 7, por fim apresenta os resultados obtidos deste trabalho, sugestões para trabalhos futuros com a linguagem computacional em Python.

2 OTIMIZAÇÃO ESTRUTURAL

Pode-se definir como otimização estrutural o processo matemático que se objetiva em obter um modelo ideal que resulta em uma performance ótima segundo algum critério de desempenho pré-estabelecido, satisfazendo algumas restrições se houver (variáveis de projeto), como custo e peso por exemplo.

A otimização estrutural está vinculada ao melhor desempenho que aquela estrutura pode oferecer, seja ao máximo carregamento que pode suportar com uma determinada seção, ou mínimo de amarrações, nós ou quantidade de barras, obtendo assim uma solução ótima para cada caso a ser estudado.

Na engenharia a otimização encontra-se diretamente relacionada a uma função $f(x)$, onde desejamos minimizar os custos e pesos mantendo o máximo do desempenho estrutural, definindo variáveis com dimensões físicas como (área da seção, largura, comprimento, assim por diante) que são chamadas de variáveis de projeto.

Fica claro que o êxito da otimização estrutural depende de alguns fatores como uma boa formulação do projeto, detalhada descrição matemática, assim como os parâmetros a serem estabelecidos juntamente com as limitações que devem estar parametrizadas (Oliveira e Falcón, 2013). Para otimizar qualquer estrutura, deve-se saber seu comportamento, seja sua propriedade geométrica ou os esforços na qual será submetida.

2.1 Estruturas Reticuladas

As estruturas reticuladas são aquelas constituídas por barras de eixo retos e articulações, sendo as principais:

- Treliças
- Vigas
- Pórticos
- Grelhas

2.1.1 Treliças

A treliça é composta por elementos com pouca espessura, geralmente dispostos de forma triangular. As treliças planas são compostas por componentes localizados no mesmo plano e geralmente são utilizadas para apoiar pontes e tetos,

enquanto as treliças espaciais possuem elementos que se estendem em três dimensões e são adequadas para guindastes e torres.

Figura 1: Pontes de treliças - sistemas de barras de aço entrelaçadas

Fonte: engwhere (2018)

2.1.2 Vigas

As vigas são geralmente membros horizontais retos, usados principalmente para suportar cargas verticais. Normalmente, são classificadas de acordo com a arquitetura, o tipo de carregamento e o material empregado (concreto, metal ou madeira), na qual muda a resistência e durabilidade. Em particular, quando a seção transversal muda, a viga é chamada de inércia variável.

Figura 2:Vigas de concreto

Fonte: cimentoholcim (2018)

2.1.3 Pórticos

As armações de pórtico são normalmente utilizadas em edifícios e consistem em vigas e colunas com ligações fixas ou de pinos. Como uma treliça, o pórtico se estende em duas ou três dimensões, suas ligações são consideradas as mais eficazes (ligações rígidas).

Figura 3: Pórtico rolante

Fonte: treicap (2017)

2.1.4 Grelhas

A grelha consiste em estruturas lineares localizadas no mesmo plano, formando uma malha que pode receber tensões que não estejam localizadas no mesmo plano. Essas hastes se cruzam e trabalham juntas para resistir a ações perpendicular a seu plano. Elas são muito utilizadas em coberturas de galpões pois resiste melhor aos esforços do vento.

Figura 4: Aplicação de concreto na malha de aço

Fonte: acomais (2020)

3 TIPOS DE OTIMIZAÇÃO ESTRUTURAL

Durante muito tempo, as construções de estruturas foram feitas através de tentativas e erro. Devido a isso, os elementos estruturais eram colossais e densos, pois na época existiam limitações de conhecimento teórico-matemáticos.

À primeira vista, o relativo sucesso e durabilidade destas estruturas pode dar uma falsa impressão de que foram bem dimensionadas. Porém, basta uma rápida comparação com as estruturas dos dias de hoje para perceber que as atuais costumam ser menores, leves e econômicas.

Figura 5: Aqueduto romano

Fonte: brasilescola (2020)

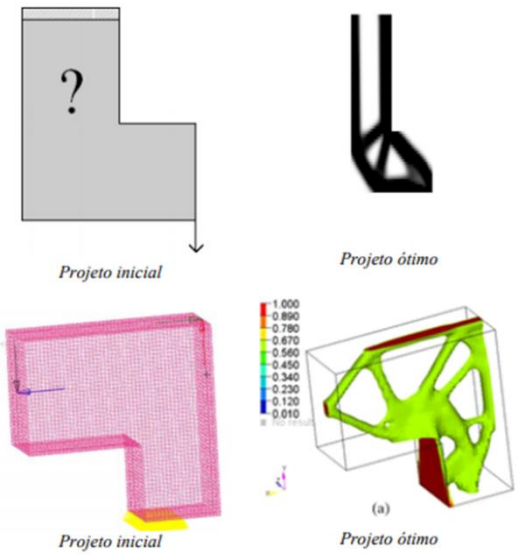
Isso foi o que impulsionou os primeiros estudos sobre otimização estrutural (MICHELL, 1904). A proposta de reduzir o peso nas estruturas que, na época, era um dos principais impedimentos na hora de projetar ou executar uma obra.

A otimização de estruturas segue três métodos, onde são listados abaixo.

3.1 Otimização Topológica

Esse tipo de otimização fundamenta-se na remoção de material da estrutura de regiões que não apresentam função estrutural considerável. A estrutura passa por um processo de aproximação discretizada que busca uma solução exata, modificando o padrão de conectividade ou a disposição espacial dos elementos e/ou membros estrutura, com vistas a um índice de performance ótimo. (Lopez e Miguel, 2013).

Figura 6: Processo de otimização topológica

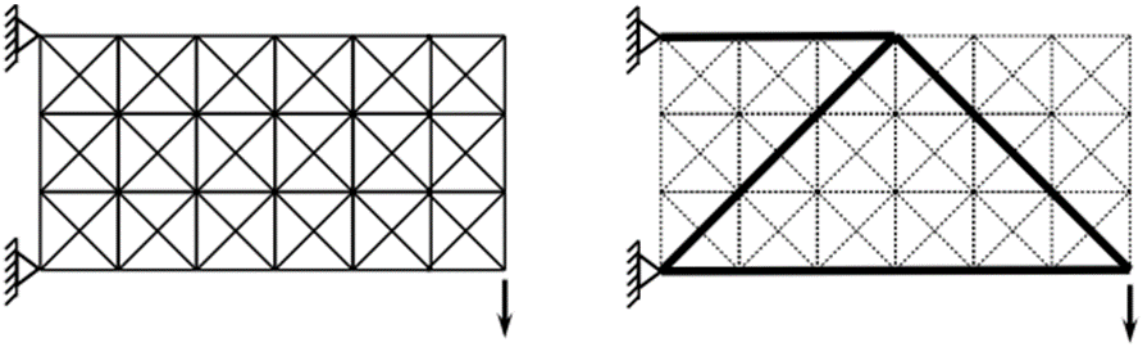


Fonte: Lopez e Miguel (2013)

Pode-se resumir então que, com o projeto inicial em mãos, o algoritmo deste método consegue apontar a retirada de material de qualquer local do projeto, minorando suas dimensões sem perder eficiência estrutural, ou seja, garantindo que a estrutura não falhe.

Neste método, tem-se a liberdade de modificar o perfil, tamanhos e conceitos geométricos.

Figura 7: Antes e depois da otimização topológica

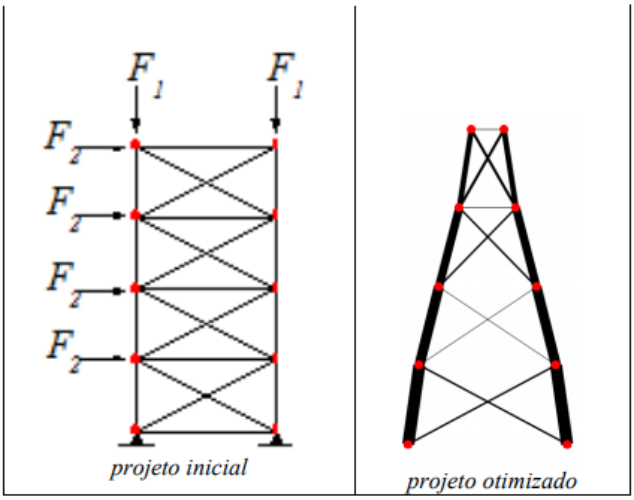


Fonte: Lopez e Miguel (2013)

3.2 Otimização de Forma ou Geométrica

Este método da otimização de geometria é utilizado na engenharia civil em projetos de treliças ou torres de transmissão. Ele busca posicionar os nós e a área da seção transversal de cada barra de forma a minimizar custos e ainda garantir que ela não falhe.

Figura 8: Processo de otimização de forma



Fonte: Lopez e Miguel (2013)

Utilizado quando o elemento estrutural já possui características pré-definidas dos componentes e se anseia em realizar um ajuste na forma, mas sem alterar as

características principais do componente, como o número de furos ou de nós em um componente por exemplo.

Esse método também pode ter aplicações em sistemas hidráulicos, aerodinâmicos e termodinâmicos, que possui como objetivo reduzir a perda de carga, arrasto e aumentar a troca térmica respectivamente.

3.3 Otimização Paramétrica

Na otimização paramétrica, o problema de otimização pode ser resolvido por uma ou mais variáveis de projeto a fim de minimizar ou maximizar uma função objetivo definida pelo engenheiro ou projetista, podendo tornar o componente que está sendo otimizado mais robusto, leve ou até mesmo reduzir os deslocamentos da estrutura.

Por exemplo, o raio da seção transversal vai ser parametrizado e variado para que possa atender os diferentes cenários e definir o modelo ideal ótimo.

Figura 9: Exemplo de otimização paramétrica

Fonte: ESSS (2020)

Este método é mais utilizado quando há limitações na modificação do perfil e da geometria da estrutura. Podendo assim ajustar por parâmetros o limite máximo que a estrutura pode ser otimizada.

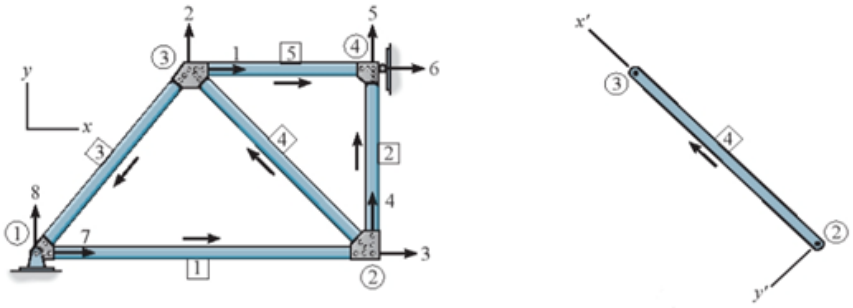
4 MÉTODO DA RIGIDEZ DIRETA APLICADO A TRELIÇAS

O método pode ser empregado em estruturas reticuladas usando valores discretos, obtidos por parâmetros que no método da rigidez direta são as forças e os momentos, ao contrário do método dos deslocamentos como o nome sugere, usa os deslocamentos ou rotações para sua análise.

A aplicação do método da rigidez requer que a estrutura seja subdividida em uma série de elementos finitos discretos como mencionado anteriormente, e seus pontos extremos sejam identificados como *nós*. Para a análise de treliça, o elemento finito é representado pelos membros (cada um deles) que constituem a treliça, e os *nós* representam a conexão entre eles. Determinando as propriedades de deslocamento de força de cada elemento, após isso definir as equações de equilíbrio de força escritas nos *nós* para relacioná-los entre si. (HIBBELER,2013).

Deve-se primeiramente montar um sistema de eixos de referência local e global.

Figura 10: Elementos de uma treliça



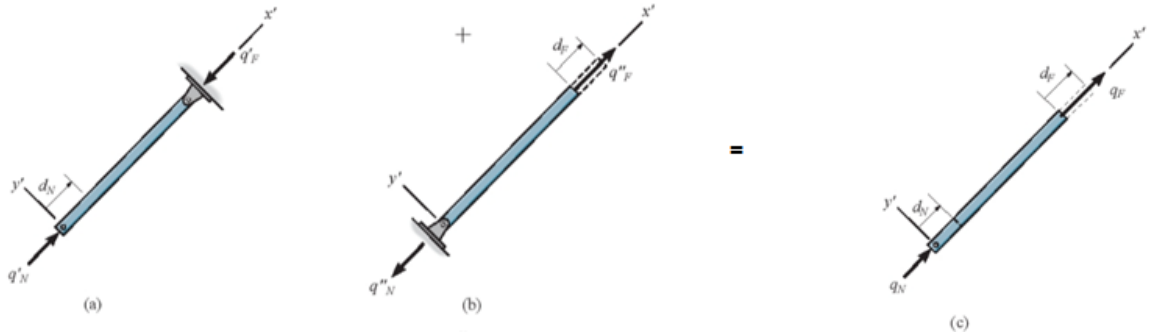
Fonte: Hibbeler (2013)

Definidos os referenciais e os elementos da matriz de rigidez dos membros, será definido os mesmos em coordenadas globais e locais, assim criando a matriz de rigidez da estrutura em si.

4.1 Matriz de rigidez do elemento

Montando um sistema de equações {F} = [K]{d}, ficará determinados os esforços de tração ou compressão em cada uma das barras. Os termos nessa matriz representarão as relações de carga-deslocamento de cada membro.

Figura 11: Superposição de forças



Fonte: Hibbeler (2013)

As forças resultantes da Figura 6, causadas por ambos deslocamentos são:

$$q_N = \frac{AE}{L} d_N - \frac{AE}{L} d_F \qquad (1)$$

$$\begin{bmatrix} q_N \\ q_F \end{bmatrix} = \frac{AE}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} d_N \\ d_F \end{bmatrix} \qquad (2)$$

Essas equações podem ser representadas na forma matricial como

$$\boldsymbol{q} = k' d \qquad (3)$$

ou

$$q_F = -\frac{AE}{L} d_N + \frac{AE}{L} d_F \qquad (4)$$

onde

$$\boldsymbol{k'} = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (5)$$

4.2 Matriz de transformação de força e deslocamento

Sabendo que uma treliça comporta muitos elementos, iremos desenvolver um método para transformar as forças dos membros $\boldsymbol{q}$ e deslocamentos $\boldsymbol{d}$ definidos por coordenadas locais e globais. Por conveniência adotaremos as coordenadas x para a direita e positiva y para cima. Os ângulos menores entre os eixos x e y positivos e os eixo local x' *positivo* serão definidos como x e y. Os cossenos dos ângulos serão usados na análise matricial. Identificados como $\lambda_x = \cos(\theta_x)$, $\lambda_y = \cos(\theta_y)$**.**

4.2.1 Matriz de transformação de deslocamento

Em coordenadas globais cada extremidade poderá ter dois graus de liberdade, assim o *nó* **N** terá D_{Nx} e D_{Ny} o nó **F** D_{Fx} e D_{Fy}, consideramos esses deslocamentos separados e obteremos seus deslocamentos ao longo do membro. Chegamos a duas expressões de deslocamentos:

$$d_N = d_{Nx}\lambda_x + D_{Ny}\lambda_y \quad (6)$$

$$d_F = d_{Fx}\lambda_x + D_{Fy}\lambda_y \quad (7)$$

Podendo ser escrita na seguinte forma matricial.

$$\begin{bmatrix} d_N \\ d_F \end{bmatrix} = \begin{bmatrix} \lambda_x & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_x & \lambda_y \end{bmatrix} \begin{bmatrix} D_{Nx} \\ D_{Ny} \\ D_{Fx} \\ D_{Fy} \end{bmatrix} \quad (8)$$

ou

$$\mathbf{d} = \mathbf{TD} \tag{9}$$

onde

$$\mathbf{T} = \begin{bmatrix} \lambda_x & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_x & \lambda_y \end{bmatrix} \tag{10}$$

Da derivação acima, $\mathbf{T}$ transforma os quatro deslocamentos x e y globais $\mathbf{D}$ nos dois deslocamentos x' locais **d**.

4.2.2 Matriz de transformação de força

Considerando agora que as forças estão atuando nas extremidades teremos as seguintes componentes de forças globais de $\boldsymbol{qn}$ em **N** são:

$$Q_{Nx} = q_N \cos\theta_x \; ; \; Q_{Ny} = q_N \cos\theta_y \tag{11}$$

De maneira semelhante as forças globais em **F** são:

$$Q_{Fx} = q_F \cos\theta_x \; ; \; Q_{Fy} = q_F \cos\theta_y \tag{12}$$

Fazendo as devidas transformações usando os cossenos diretores, teremos que estas equações tornam-se

$$\begin{aligned} Q_{Nx} &= q_N\lambda_x \;\; ; \;\; Q_{Ny} = q_N\lambda_y \\ Q_{Fx} &= q_F\lambda_x \;\; ; \;\; Q_{Fy} = q_F\lambda_y \end{aligned} \tag{13}$$

que são escritas em forma matricial como

$$\begin{bmatrix} Q_{Nx} \\ Q_{Ny} \\ Q_{Fx} \\ Q_{Fy} \end{bmatrix} = \begin{bmatrix} \lambda_x & 0 \\ \lambda_y & 0 \\ 0 & \lambda_x \\ 0 & \lambda_y \end{bmatrix} \begin{bmatrix} q_N \\ q_F \end{bmatrix} \tag{14}$$

ou

$$\mathbf{Q} = \mathbf{T}^T\mathbf{q} \tag{15}$$

onde

$$\mathbf{T}^T = \begin{bmatrix} \lambda_x & 0 \\ \lambda_y & 0 \\ 0 & \lambda_x \\ 0 & \lambda_y \end{bmatrix} \tag{16}$$

Com isso $\mathbf{T}^T$ transforma as duas forças(x') locais **q** atuantes na extremidade do membro em quatro componentes de força (x, y) globais **Q**.

Por fim combinamos os resultados das seções anteriores para determinar a matriz de rigidez global do elemento. Relacionando as forças globais dos elementos com seus respectivos deslocamentos, obtemos a seguinte equação:

$$\mathbf{q} = \mathbf{k'TD} \tag{17}$$

Substituindo esta equação na Equação (17), $\boldsymbol{Q = T^T q}$**,** produzindo o resultado final,

$$\mathbf{Q} = \mathbf{T}^T\mathbf{k'TD} \tag{18}$$

ou

$$\mathbf{Q} = \mathbf{kD} \tag{19}$$

onde

$$\mathbf{k} = \mathbf{T}^T\mathbf{k'T} \tag{20}$$

A matriz **k** é a matriz de rigidez global do membro em coordenadas globais tendo visto que sua matriz de transformação e a transposta, juntamente com matriz de rigidez local são conhecidos, logo temos.

$$\mathbf{k} = \begin{bmatrix} \lambda_x & 0 \\ \lambda_y & 0 \\ 0 & \lambda_x \\ 0 & \lambda_y \end{bmatrix} \frac{AE}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \lambda_x & \lambda_y & 0 & 0 \\ 0 & 0 & \lambda_x & \lambda_y \end{bmatrix} \tag{21}$$

Realizando as operações de matriz resulta em

$$\mathbf{k} = \frac{AE}{L} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} N_x & N_y & F_x & F_y \end{matrix} \begin{bmatrix} \lambda_x^2 & \lambda_x\lambda_y & -\lambda_x^2 & -\lambda_x\lambda_y \\ \lambda_x\lambda_y & \lambda_y^2 & -\lambda_x\lambda_y & -\lambda_y^2 \\ -\lambda_x^2 & -\lambda_x\lambda_y & \lambda_x^2 & \lambda_x\lambda_y \\ -\lambda_x\lambda_y & -\lambda_y^2 & \lambda_x\lambda_y & \lambda_y^2 \end{bmatrix} \begin{matrix} N_x \\ N_y \\ F_x \\ F_y \end{matrix} \tag{22}$$

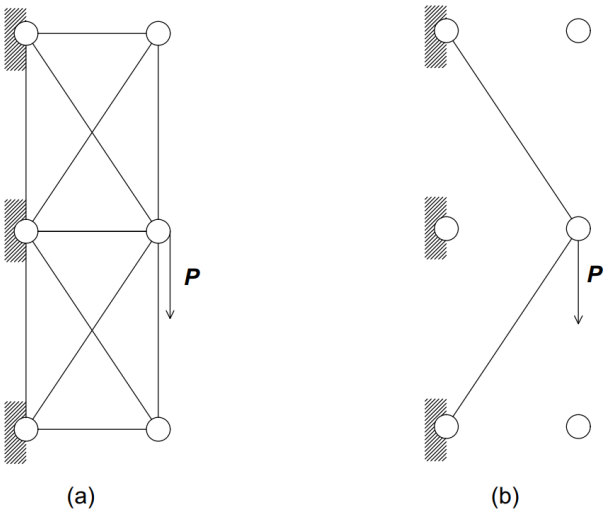
A localização de cada elemento desta matriz é simétrica 4 x 4, referenciando os graus de liberdade associados às extremidades N e F. Para mais detalhes do método de rigidez global aplicado em treliça ler Análise Estrutural de Hibbeler 2013.

5 ROTINAS COMPUTACIONAIS PARA A OTIMIZAÇÃO DE TRELIÇA

O objetivo é projetar a melhor estrutura possível, escolhendo as áreas da seção transversal dos membros da treliça. A área da seção transversal de uma barra elástica é uma variável natural de espessura da estrutura. Na literatura esse problema se encaixa em problemas de otimização de dimensionamento, uma vez que estamos otimizando o tamanho dos elementos estruturais da treliça; entretanto, se permitimos que essas áreas tomem o valor zero no problema de otimização, então isso passa a se caracterizar como otimização topológica, pois quando isso acontece, as barras são removidas da estrutura como representado na Figura 12 (b). Os critérios a serem otimizados e as restrições para o problema podem ser de qualquer tipo com significado físico. Entre as mais usadas na engenharia, está a de maximizar a rigidez da estrutura mantendo a restrição de volume. É claro que o volume máximo permitido

é pequeno o suficiente para obter uma estrutura com apenas duas barras com mostra a Figura 12 (b).

Figura 12: Exemplo de uma treliça da esquerda para direita antes e depois da otimização



Fonte: Adaptado de Aranda e Bellido (2013)

5.1 Problemas de otimização

Assume-se que as barras terão comportamento linear elástico. Por conveniência, admite-se que a treliça do estudo é feita de barras de mesmo material com o módulo Young constante E, embora diferentes materiais possam ser considerados no algoritmo.

Ao unir as áreas da seção transversal de todas as barras a serem otimizadas em um vetor $\mathbf{X} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$, (será utilizado caracteres em negrito para vetores e matrizes) sendo *n* número dessas barras, então o problema de otimização tem a seguinte forma.

$$\underset{X \epsilon U_{ad}}{\text{Minimize}}\ C(x) = \mathbf{F}^T\mathbf{U} \tag{23}$$

sujeito a

$$\mathbf{K}(\mathrm{x})\mathbf{U} = \mathbf{F} \tag{24}$$

e a restrição de volume

$$\sum_{j=1}^{n} l_i x_j \leq V_{\max.} \tag{25}$$

O conjunto viável para o problema de otimização, U_{ad} é dado por:

$U_{ad} = \{\boldsymbol{x} = \in \mathbb{R}^n : x_j^{min} \leq x_j \leq x_j^{max}, x = 1, \dots, n\}, x_j^{min}, x_j^{max}$ respectivamente, as áreas de seção transversal máxima e mínima para j- barra. Chamaremos m de número de *nós* cujo o deslocamento não é restringido pelas condições de contorno. Então $\mathbf{U} \in \mathbb{R}^{2m}$ representa o vetor de deslocamento e $\mathbf{F} \in \mathbb{R}^{2m}$ representa o vetor de cargas aplicadas nos *nós* da estrutura. Observando que nosso modelo é bidimensional portanto, se o número de *nós* livres for m então os componentes do vetor de deslocamento e carga deve ser $2m$. O custo funcional $C(x) = \mathbf{F^T U}$, é a conformidade, ou trabalho realizado pela carga para deformar a estrutura. A equação (...), onde $\mathbf{K(x)}$ representa a matriz de rigidez direta da estrutura. Minimizar a função $C(x)$ equivale a maximizar a matriz de rigidez pois, quanto menor for sua conformidade mais rígida é a treliça.

5.1.1 Análise matemática

A seguir será abordado questões analíticas sobre o problema de otimização da estrutura. Para isso admite-se que a matriz de rigidez é positiva e definida para qualquer projeto admissível em x, logo para qualquer $x \in U_{ad}$ haverá uma solução única para o sistema.

$$\mathbf{K}(\mathrm{x})\mathbf{U} = \mathbf{F},$$

que denotemos o $\mathbf{U}(\mathrm{x})$. Portanto a formulação alinhada desse problema é

$$\underset{X \epsilon U_{ad}}{\text{Minimize}}\ C(x) = \mathbf{F}^T \mathbf{U}\,(x) \tag{26}$$

sujeito a

$$\sum_{j=1}^{n} l_i x_j \leq V_{\max.} \tag{27}$$

Esse é um problema de otimização não-linear. As restrições de caixa (aquelas que definem o conjunto admissível) e uma restrição linear (restrição de volume). A

existência de soluções ótimas pode ser mostrada de outras formas, mas para um problema de dimensões finitas como este, provavelmente o mais fácil é perceber que o conjunto viável para o problema de otimização, U_{ad} é um subconjunto compacto de $\mathbb{R}^n$ e que o custo funcional $C(x)$ é continuo nesse conjunto.

Para o algoritmo numérico, se faz necessário o uso das derivadas do custo funcional C com respeito das variáveis de projeto x_j. Isso é obtido de maneira direta.

$$\frac{\partial C(x)}{\partial x_j} = \frac{\mathbf{F}^T \partial \mathrm{U}(\mathrm{x})}{\partial x_j} = \left(\mathbf{K}(\mathrm{x})\mathbf{U}(\mathrm{x})\right)^T \frac{\partial \mathrm{U}(\mathrm{x})}{\partial x_j} = \mathbf{U}(\mathrm{x})^T \mathbf{K}(\mathrm{x}) \frac{\partial \mathrm{U}(\mathrm{x})}{\partial x_j} \tag{28}$$

se agora derivarmos parcialmente a equação, obteremos

$$\frac{\partial \mathbf{K}(\mathrm{x})}{\partial x_j}\mathbf{U}(\mathrm{x}) + \mathbf{K}(\mathrm{x})\frac{\partial \mathbf{U}(\mathrm{x})}{\partial x_j} = 0,$$

e depois

$$\frac{\partial \mathbf{U}(\mathrm{x})}{\partial x_j} = -\mathbf{K}(\mathrm{x})^{-1}\frac{\partial \mathbf{K}(\mathrm{x})}{\partial x_j}\mathbf{U}(\mathrm{x}) \tag{29}$$

finalmente substituímos a equação (29) na (28) e obtemos.

$$\frac{\partial C(x)}{\partial x_j} = -\mathbf{U}(\mathrm{x})^T \frac{\partial \mathbf{K}(\mathrm{x})}{\partial x_j}\mathbf{U}(\mathrm{x}) \tag{30}$$

A derivada de matriz de rigidez em relação a x_j é muito fácil de obter já que x_j só aparece em $\mathbf{K}(\mathrm{x})$ multiplicando a rigidez elementar $\boldsymbol{k}_j$ (nas posições onde está matriz elementar ocupa quando $\mathbf{K}(\mathrm{x})$ é montado), de modo que

$$\frac{\partial C(x)}{\partial x_j} = -\boldsymbol{u}_j^T \boldsymbol{k}_j \boldsymbol{u}_j \tag{31}$$

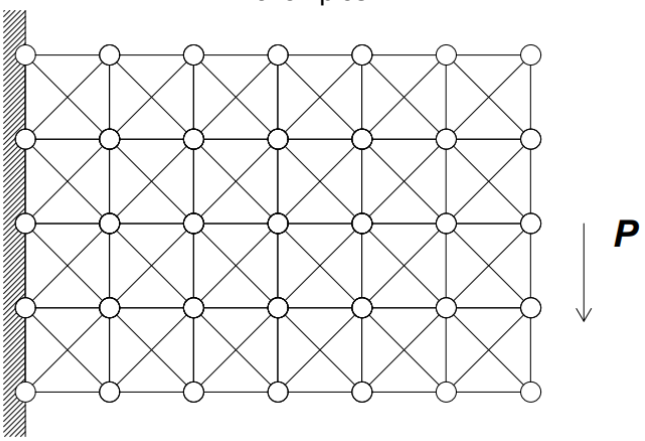
5.2 Implementação

Com o auxílio da linguagem computacional Python para a implementação do código, inspirado na idealização de Aranda e Bellido (2013) criadores do código principal. Apesar do Python ainda não ser amplamente utilizado na comunidade de

engenheiros e matemáticos, ela é uma linguagem muito poderosa, legível e fácil de aprender. A utilização de métodos computacionais brevemente se tornará um ambiente padrão na engenharia, devido a sua praticidade e comodidade em elaborar roteiros de cálculos e pré-dimensionamento e dimensionamento de estruturas.

Essa implementação é dividida em duas etapas. Na primeira, será construída uma estrutura de treliça uniforme com os *nós* e *barras*, e na segunda etapa, o código para otimização estrutural é criado.

Figura 13: Estrutura de treliça e carga para os exemplos.



Fonte: Adaptado de Aranda e Bellido

5.2.1 Criação da estrutura de treliça

As informações relevantes em uma estrutura reticulada de treliça são obtidas pelas coordenadas dos *nós* e pelas *barras* que os conectam. Para construir uma estrutura uniforme como na Figura 13, definimos um retângulo dado por seus cantos inferior esquerdo e superior direito (cada um deles é uma tupla de dois elementos para as coordenadas) e o número de *nós* usado em cada direção (dado por dois inteiros); esses são parâmetros usados na função **meshtruss**. A função retorna duas ***arrays*:** um com as coordenadas dos *nós* da estrutura e outro com as *barras* de conexão.

Por exemplo, a estrutura simples da Figura 15 corresponde ao seguinte script:

```
>>> coord, connec = meshtruss ((0,0), (1,1), 1, 1)
```

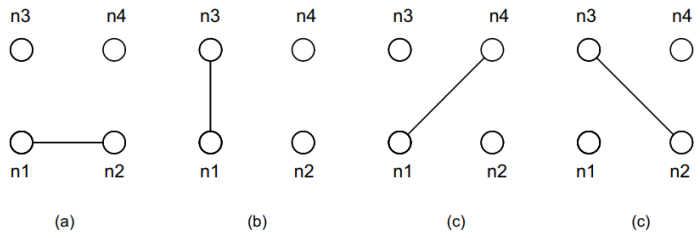
e as matrizes obtidas são:

```
>>> coord
array (  [ [ 0., 0.],
           [ 1., 0.],
           [ 0., 1.],
           [ 1., 1.] ] )
>>> connec
array ( [ [ 0, 1 ],
          [ 0, 2 ],
          [ 0, 3 ],
          [ 1, 2 ],
          [ 1, 3 ],
          [ 2, 3 ] ] )
```

De modo que [coord[i, 0], coord[i, 1]] são as coordenadas do *nó* – i e connec[j, 0] e connec[j, 1] são índices dos *nós* no fim da *barra* – j.

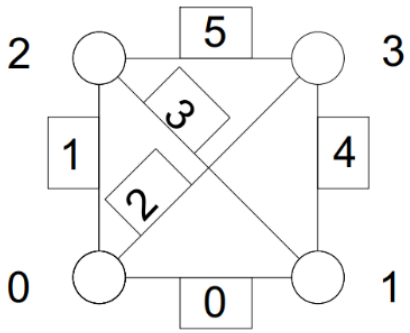
As configurações dos nós e das barras serão definidas de conforme o script acima estão representadas nas figuras abaixo.

Figura 14: Criação de barra em uma estrutura uniforme



Fonte: Adaptado de Aranda e Bellido (2013)

Figura 15: Estrutura básica de uma treliça



Fonte: Adaptado de Aranda e Bellido (2013)

6 RESULTADOS DA OTIMIZAÇÃO DA ESTRUTURA DE TRELIÇA

O código principal para a otimização da estrutura foi implementado através da função **Opttruss**, que permite resolver a equação (26) usando o algoritmo método de mover assíntotas (MMA), desenvolvido por (Svanberg, 1987), que necessita de que o usuário forneça a função de custo, as restrições e seus derivados correspondentes. Os códigos apresentados neste trabalho foram idealizados por (Arranda e Bellido, 2013), porém atualmente, o algoritmo (MMA) se encontra indisponível no repositório do Python, sendo necessário utilizar outro algoritmo de solução, a programação sequencial de quadrados mínimos (SLSQP), este algoritmo substitui perfeitamente o MMA, solucionando o problema de otimização.

Os parâmetros de entrada desta função são as informações da estrutura, dada pelas coordenadas e conexões dos *nós* e *barras*, como foi calculado no item 5.2.1, e algumas informações adicionais sobre as condições da estrutura, como: o modulo Young de cada barra, os *nós* que são (parcialmente) fixos, as forças externas atuando em cada nó, e um parâmetro correspondente a normalização do $\boldsymbol{V_{max}}$ (27). Também existe um parâmetro padrão chamado boolean (onde admite dois valores), que estabelecem se os deslocamentos na estrutura devem ser plotados.

As funções **fobj** e **volume** calculam a função objetivo e a restrição de volume, respectivamente, junto com as suas derivações; e a função **drawtruss** permite o desenho da estrutura. Para os procedimentos de otimização iremos utilizar 3 códigos:

Meshtruss – onde cria a treliça uniforme com que servirá como parâmetro para os exemplos abordados nesse trabalho.

Opttruss – otimiza a treliça com os dados de outras funções como, *fobj,volume,drawtruss*.

Remove – remove barras e nós, específicos da treliça.

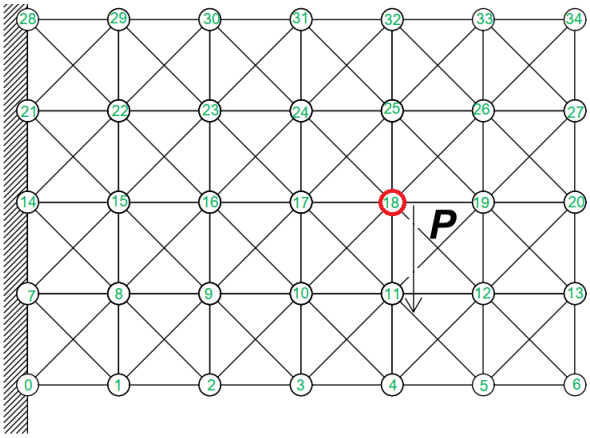
Todos os códigos e scripts usados para esta otimização, assim como o procedimento de instalação estará localizado no apêndice deste trabalho.

6.1 Exemplo 1 – Otimização da treliça uniforme

O processo de otimização da será resolvido por meio da função **NLP** do módulo *Openopt* (Kroshko,2007), onde a mesma necessita da função de custo e seus derivados, as restrições e seus derivados, todos esses valores foram obtidos das funções **fobj** e **volume** já mencionadas.

O carregamento pontual estará localizado no *nó* 18 conforme a Figura 16, o programa irá otimizar a estrutura para esse carregamento.

Figura 16: Carregamento pontual aplicado no nó 18



Fonte: Autor

Por fim implementada a função **drawtruss** para desenhar a estrutura ideal e deslocamentos, realizou-se um loop em todas as *barras* da estrutura e construímos as barras originais e as barras deslocadas, além disso modificamos a **linewidth** (função que modifica a espessura da linha). A cor na Figura 18, reflete se a barra está trabalhando sob tensão (vermelho) ou compressão (azul).

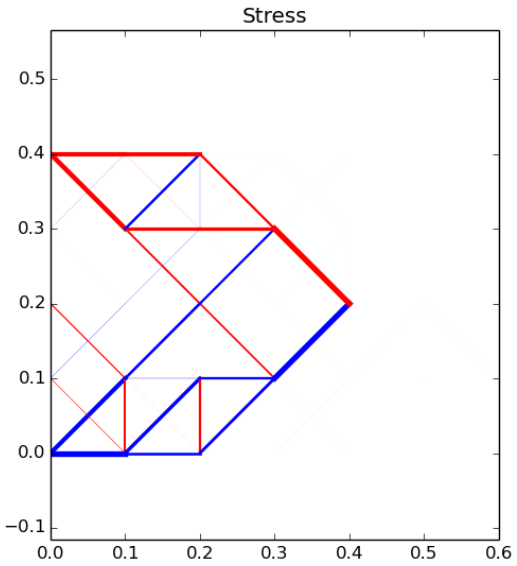
Ao iniciar a programação, o script mostra interface de usuário para configurar o problema da estrutura, no exemplo será considerado que todas as barras são feitas do mesmo material e há apenas uma carga vertical localizada no nó 18.

Figura 17: Interface de usuário

```
------------------------ OpenOpt 0.38 -------------------------
solver: scipy_slsqp   problem: Truss    type: NLP   goal: minimum
 iter   objFunVal   log10(maxResidual)
    0  3.323e+00            -100.00
   80  1.153e-01             -14.24
istop: 1000 (Optimization terminated successfully.)
Solver:   Time Elapsed = 4.7     CPU Time Elapsed = 4.707381
objFunValue: 0.11528713 (feasible, MaxResidual = 5.79186e-15)
```

Fonte: Autor

Figura 18: Otimização da treliça do exemplo 1

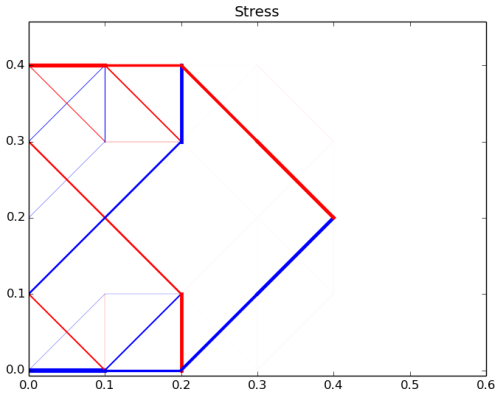


Fonte: Autor

Fica constatado que algumas barras estão com sua seção transversal mais espessas que outras, para que possa resistir aos esforços do carregamento aplicado no *nó* 18. Nota-se que o que o *nó* 16 é serve como interseção das barras (16-22), (16-10) e (16-24), (16-8).

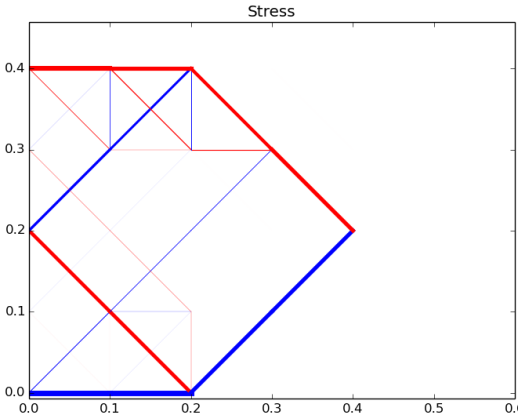
Para demonstrar a capacidade de otimização topológica, foi aplicado um script no qual usará a função **remove**, para retirar o *nó* 16 e será feito o mesmo com a barra (16-22).

Figura 19: Resultado sem nó 16



Fonte: Autor

Figura 20: Resultado sem a barra 16-22



Fonte: Autor

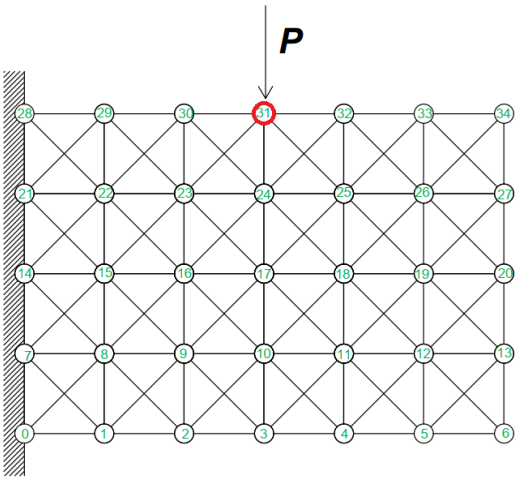
Como pode ser visto na Figura 19, com a retirada do nó 16, os esforços foram transferidos para o nó 15, mantendo o equilíbrio da treliça. Já com a remoção da barra 16-22, foi criado duas barras paralelas para suportar os esforços transmitidos pelo carregamento.

6.2 Exemplo 2 – Otimização de uma treliça uniforme com carregamento centralizado na malha.

O carregamento será aplicado no *nó* 31, e com isso a será moldada um modelo ótimo perfeito para o suporta o carregamento, iremos manter o engate por todo o lado esquerdo da estrutura.

A imagem abaixo mostra a localização do carregamento exercido na treliça

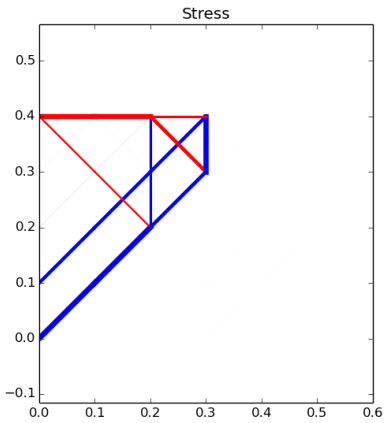
Figura 21: Carregamento no nó 31



Fonte: Autor

Ao ser aplicado o carregamento no programa, gerou-se a seguinte treliça otimizada.

Figura 22: Resultado do exemplo 2



Fonte: Autor

Nota-se que a carga foi distribuída para os apoios e que tensionando as barras superiores e comprimindo as barras inferiores.

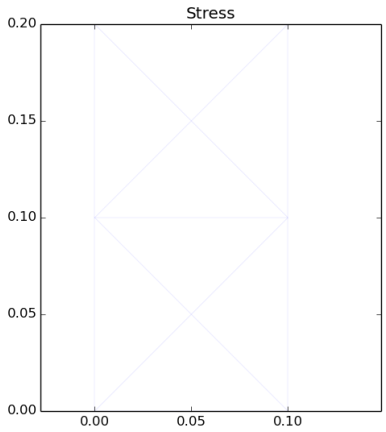
6.3 Exemplo 3 – Otimização de uma treliça com malha (1x2)

Nesse exemplo é mudada a malha da treliça uniforme usando a função **meshtruss**, mudando suas coordenadas para as seguintes:

```
>>> coord, connec = meshtruss ((0,0), (0.1,2), 1, 2)
```

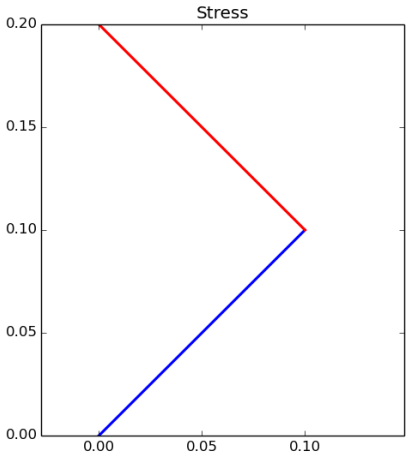
Com essa modificação, altera-se o formato da malha de treliça uniforme para uma treliça simples igual a treliça da Figura 12, vinculando o engastamento no lado esquerdo.

Figura 23: Malha (1X2)



Fonte: Autor

Figura 24: Resultado do exemplo 3



Fonte: Autor

7 CONSIDERAÇÕES FINAIS

O trabalho tem como objetivo central, otimizar estruturas reticuladas em especial treliças, usando linguagem computacional Python. Foram analisados alguns modelos de treliças, alterando sua malha e carregamentos nos *nós* e *barras*.

Numa concepção geral sobre analise estrutural há fatores que influenciam diretamente o processo de otimização, como sua seção transversal e restrições normativas afim de garantir o bom funcionamento da estrutura. Baseando-se no Método da Rigidez Direta, que é um método que possui grande confiabilidade nos cálculos estruturais, esse estudo conseguiu otimizar com sucesso todos os exemplos de nele proposto. Visto que, o procedimento realizado aplica-se a treliças estáticas planas, com algumas alterações nos códigos é possível ser utilizado em outras estruturas reticuladas como, pórticos, vigas e grelhas somente para verificação axial. Este método usa otimização topológica e alguns parâmetros para solucionar o problema de excesso de material presente na estrutura, sendo eficaz nas distribuições dos esforços nele aplicado.

As estruturas aqui calculadas foram analisadas em termo de segurança e módulo de elasticidade igual em todas as barras e nós das estruturas vistas, mostrando que esse roteiro se aplica apenas a estruturas de mesmo material. Para pesquisas futuras é recomendado que seja calculado barras com materiais diferentes e uma análise detalhada sobre a economia de material para projetar uma estrutura.

REFERÊNCIAS BIBLIOGRÁFICAS

ARANDA, Ernesto; BELLIDO, José C. Introduction to Truss Structures Optimization with Python. **The Electronic Journal of Mathematics and Technology**, Universidad de Castilla - La Mancha Spain, p. 1-17, 2013.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR-6120. **Ações para o cálculo de estruturas de edificações**. Rio de Janeiro, 1978.

BELEGUNDU, Ashok D; CHANDRUPATLA, Turupathi R. **Elementos Finitos**. 1. Ed. São Paulo: Pearson, 2014. p. 118-127.

HIBBELER, R.C. Análise de treliça usando o método da rigidez. In: **Análise das Estruturas**. 8. ed. [S. l.]: Pearson, 2013. cap. 14, p. 403-407.

LOPEZ , Rafael ; MIGUEL, Leandro F. INTRODUÇÃO A OTIMIZAÇÃO ESTRUTURAL. **Programa De Pós Graduação Em Engenharia Civil**, [*S. l.*], p. 1-7, maio 2013.

MICHELL, A. L. The limits of economy of material in frame-structures. **Philosophical Magazine Series 6**, v.8, n.47, p.589–597, 1904.

O Que é Otimização Estrutural? [*S. l.*], 27 out. 2019. Disponível em: https://www.otmza.com.br/otimizacao-estrutural-aplicacoes/. Acesso em: 3 nov. 2020.

SILVA, F. B. (2011). Algoritmos Genéticos para Otimização de Estruturas Reticuladas Baseadas em Modelos Adaptativos e Lagrangeano Aumentado. **Dissertação de mestrado, Universidade Federal de Juiz de Fora**, Juiz de Fora,186 p.

SILVA, F. E. CASTRO. **Otimização Dimensional, De Forma E Topológica De Estruturas Treliçadas Utilizando Um Algoritmo Híbrido**. 2015. Dissertação (Mestra em Engenharia Mecânica do Curso de Mestrado do Programa de Pós-Graduação em Engenharia Mecânica) - Universidade Federal do Paraná, [*S. l.*], 2015.

PIZZIRANI, F. (2003). Otimização Topológica de Estruturas Utilizando Algoritmos Genéticos. **Dissertação de mestrado, Universidade Estadual de Campinas**, São Paulo,104 p.

RIBEIRO, Leandro S. ; RODRIGUES, Eliana C.; SILVA, Lucas F.; JUNIOR, Pedro M. Aplicação Do Método Da Rigidez Direta Na Análise Matricial De Treliças Planas Indeterminadas Estaticamente. **Semana de Educação, Ciência e Tecnologia - SECITEC**, [*S. l.*], p. 2-4, 17 out. 2017.

APÊNDICE A – Montando o ambiente de desenvolvimento

Antes de começar a codificar e analisar os resultados, foi necessário montar o ambiente de desenvolvimento no computador.

Pensando em facilitar a instalação dos pacotes e das bibliotecas que seriam utilizadas durante a pesquisa, utilizamos uma plataforma muito conhecida pela comunidade chamada Anaconda. Ela contém ferramentas essenciais como o Spyder, além de um vasto conjunto de pacotes populares para cálculos matemáticos e ciência de dados que já são adicionados durante sua instalação.

A ideia aqui é codificar utilizando o Spyder, uma IDE dedicada à comunidade matemática, que já possui integração nativa com o interpretador IPython e com os pacotes numpy, scipy e matlibplot.

1.1. Instalando a máquina virtual

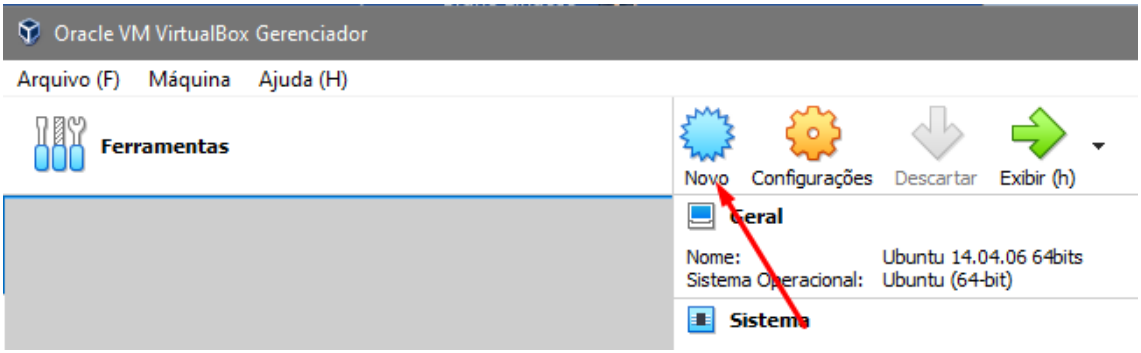
Para facilitar a instalação e configuração do ambiente de pesquisa, se fez necessário encontrar um repositório que pudesse baixar os pacotes como o openopt e o nlopt, o Spyder, além de uma versão antiga do Anaconda.

O sistema operacional Linux Ubuntu 14.04 LTS possui em seu repositório todos os estes pacotes, tornando assim seu download e instalação mais simples.

Como o sistema que utilizo por padrão é o Windows 10, foi preciso criar uma máquina virtual utilizando a ferramenta Virtual Box para emular o sistema Linux e configurá-lo de acordo com os requisitos necessários.

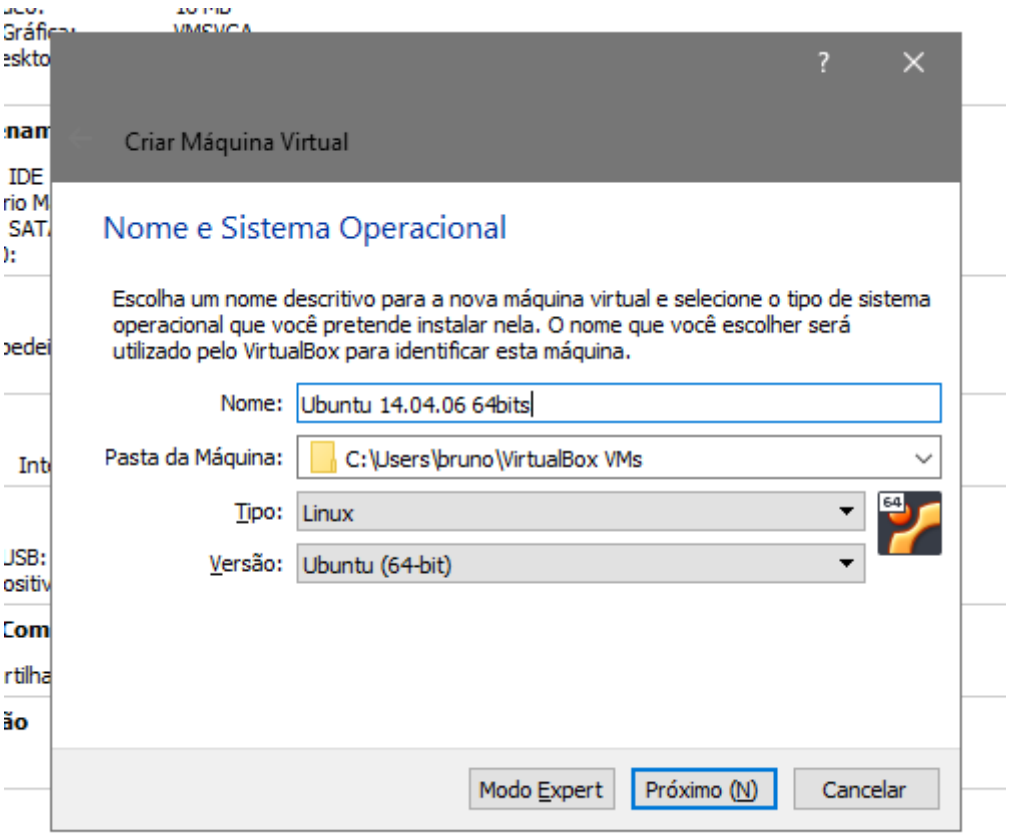
Para realizar o download do VirtualBox, acesse o link (https://www.virtualbox.org/wiki/Downloads) e escolha a versão para o seu sistema operacional hospedeiro (no meu caso o Windows 10).

Depois de instalar o VirtualBox, vamos para a etapa de criação da máquina virtual. Ao iniciar o VirtualBox, clique em Novo.



Fonte: Autor

Defina aqui o nome da máquina e qual o sistema operacional que será instalado.



Fonte: Autor

Recursos alocados na máquina virtual:

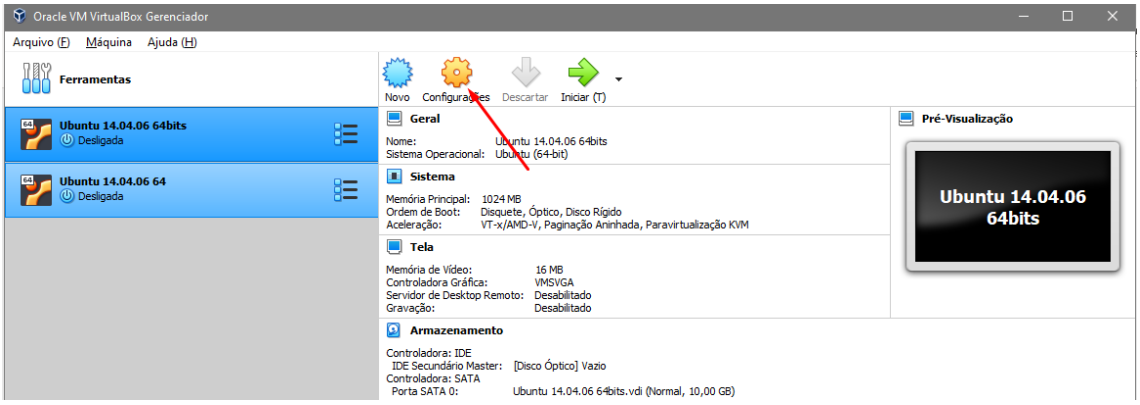
- 1GB de memória RAM
- 10GB de disco rígido virtual pré-alocado do tipo VDI (VirtualBox Disk Image)

- Disco dinamicamente alocado
- 1 CPU

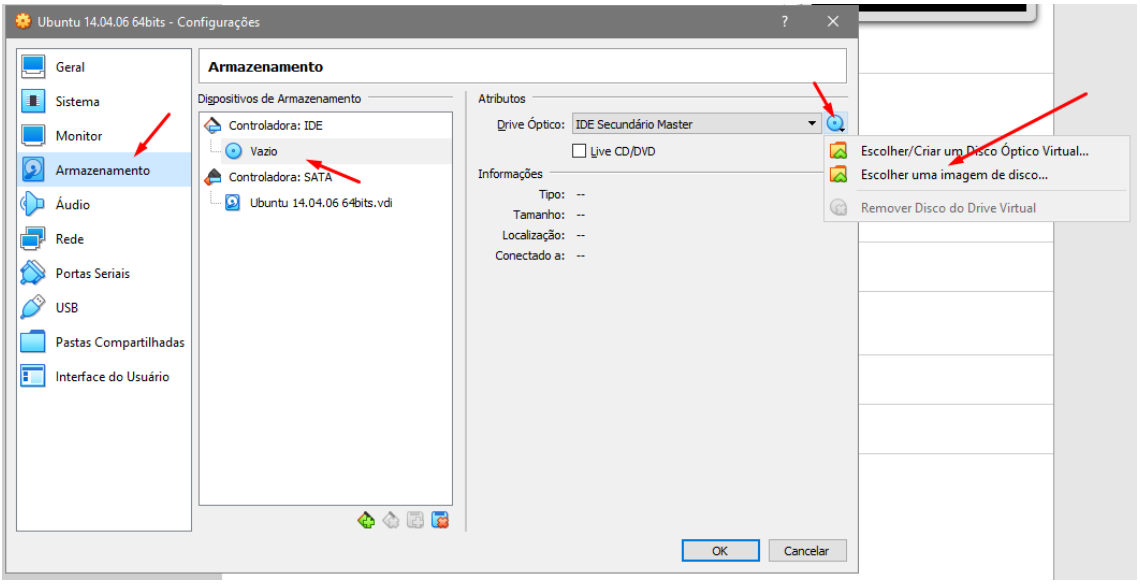
Após a criação da máquina virtual, adiciona-se uma imagem de disco do Ubuntu no formato .iso na controladora IDE, para que a máquina consiga dar o BOOT por ela e inicie a instalação do Linux.

Para baixar a imagem de disco do Ubuntu na versão 14.04, acesse (https://releases.ubuntu.com/14.04/).

Em seguida, clique em Configurações > Armazenamento > Controladora IDE > Vazio > Escolher uma imagem de disco.

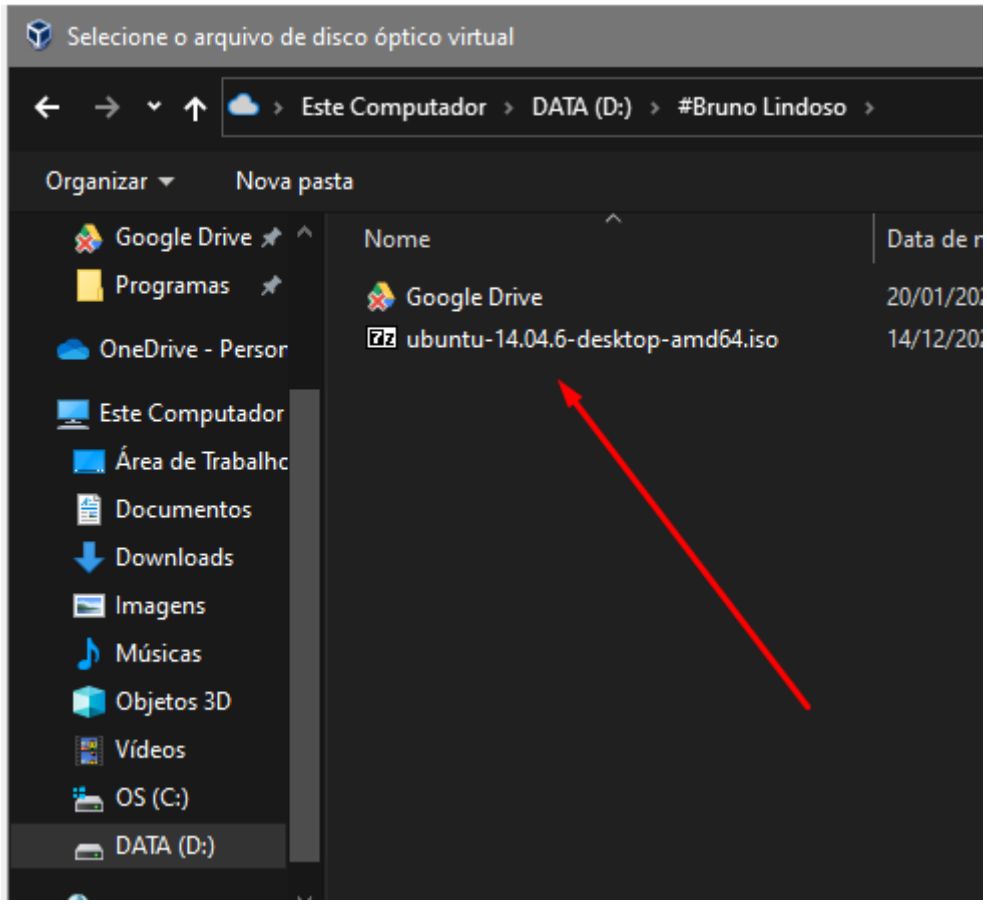


Fonte: Autor



Fonte: Autor

Selecione a imagem de disco.

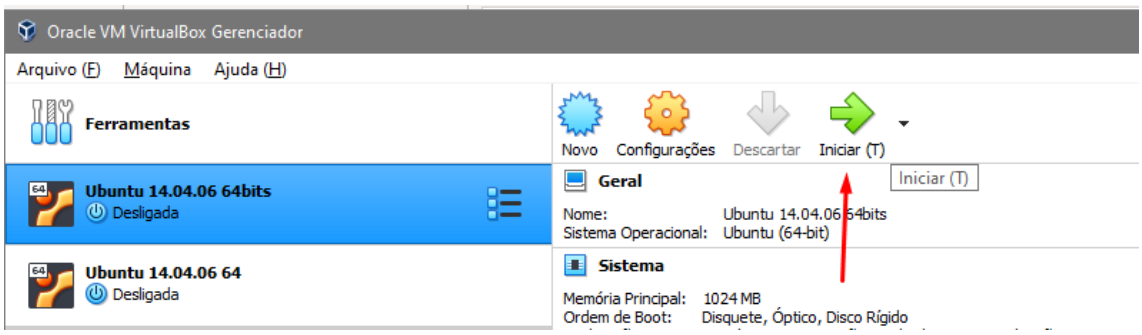


Fonte: Autor

Pronto, agora a máquina virtual está pronta para ser utilizada.

1.2. Instalando o sistema operacional Linux Ubuntu 14.04 LTS

Primeiro, inicia-se a máquina virtual no VirtualBox, clicando no botão Iniciar.

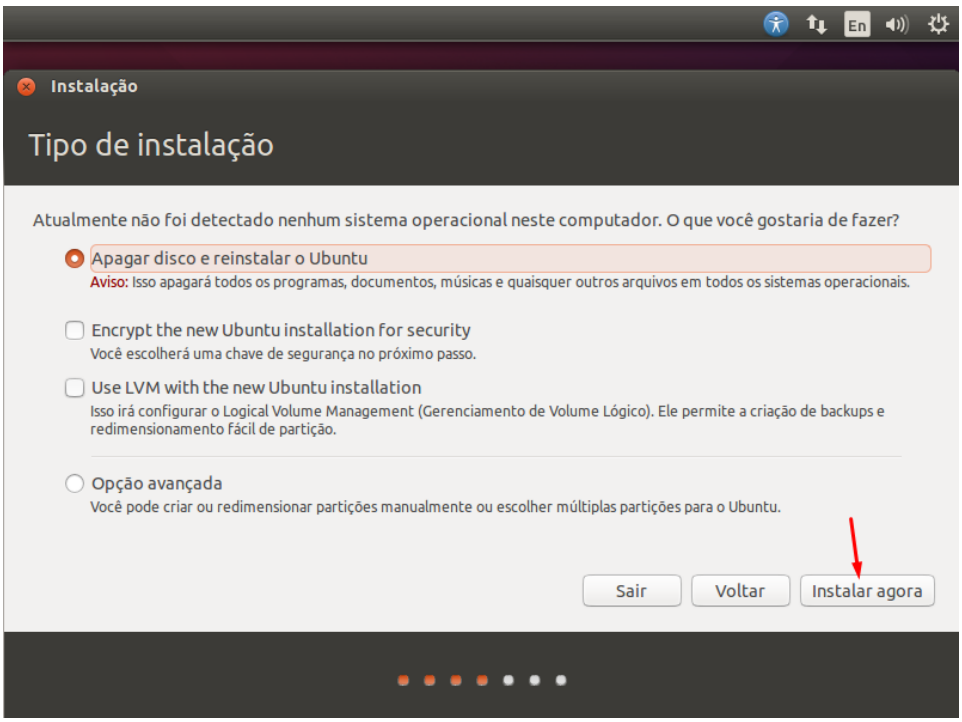


Fonte: Autor

Ele irá dar BOOT através da imagem de disco que foi definida na controladora IDE. Assim que o disco foi iniciado, ele apresenta a tela a seguir. Selecione o idioma português do Brasil e clique em Instalar o Ubuntu.

Fonte: Autor

Na proxima etapa, selecoine Apagar disco e reinstalar o ubuntu e clique em Instalar agora.



Fonte: Autor

Em seguida, confirme a formatação e a instalação do sistema de arquivo, defina a sua localidade (para configurações de fuso horário), as configurações do teclado, o nome de usuário e senha.

Pronto, depois disso inicia a cópia dos arquivos e a instalação do sistema operacional na máquina virtual.

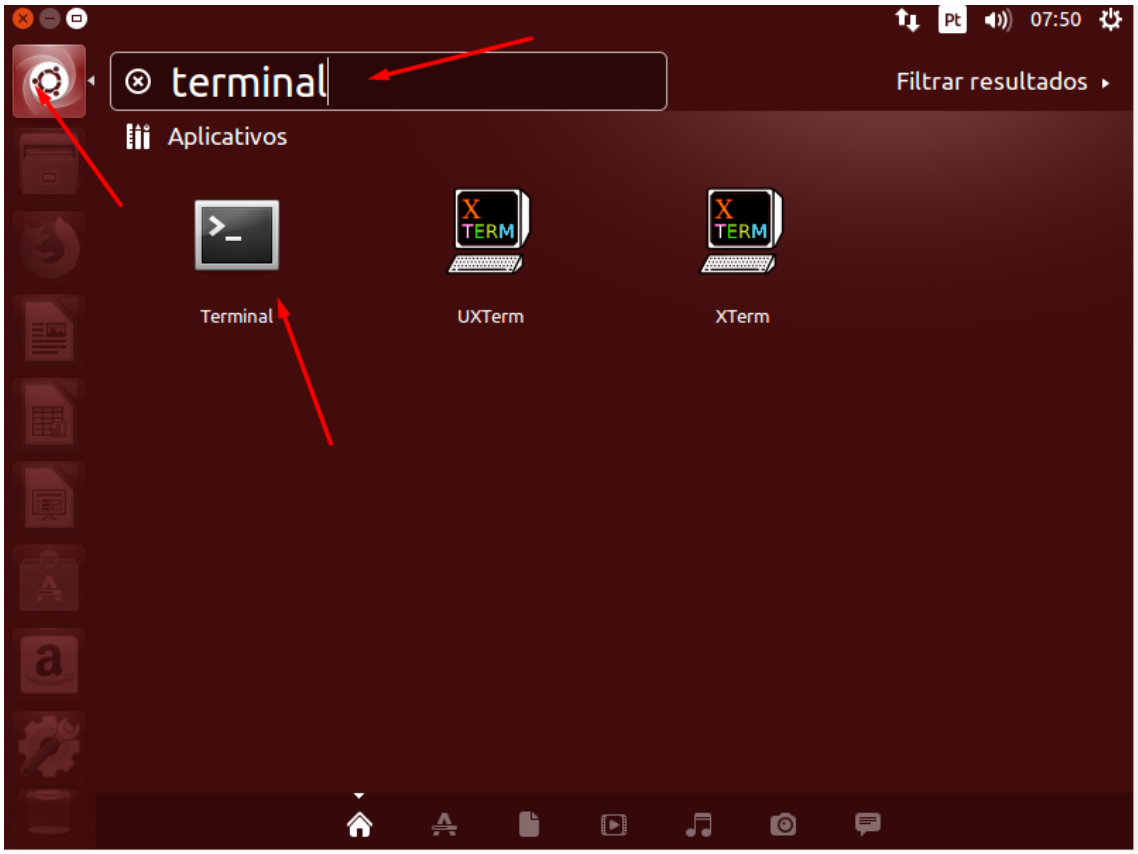
Fonte: Autor

Aguarde a instalação concluir e reinicia a máquina virtual.

1.3. Instalando os pacotes adicionais

Depois da máquina virtual instalada, instalamos os pacotes Anaconda, Spyder, openopt e nlopt

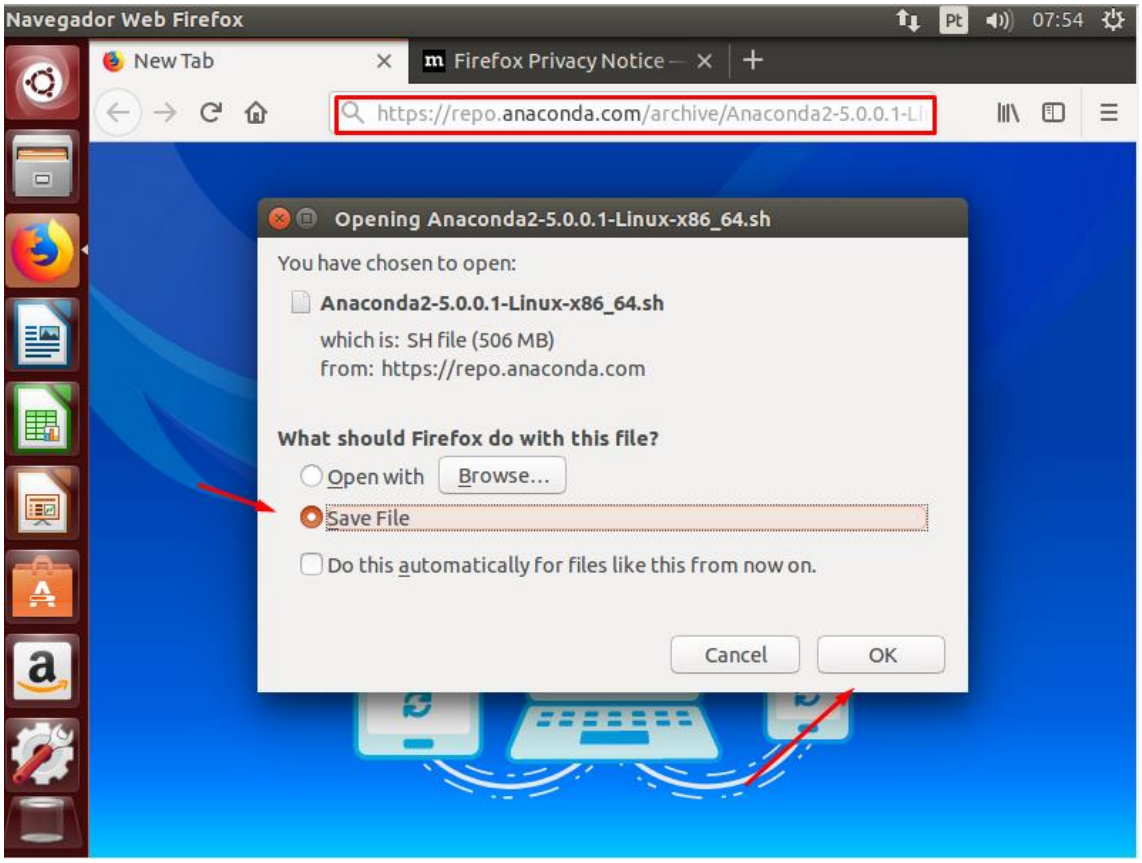
Já com o Ubuntu iniciado, acesse o terminal clicando no ícone do Ubuntu > digite Terminal > clique em Terminal.



Fonte: Autor

1.3.1. Instalando o Anaconda 2

Para baixar o Anaconda, entre no repositório através do link (https://repo.anaconda.com/archive/Anaconda2-5.0.0.1-Linux-x86_64.sh) e realize seu download.



Fonte: Autor

No terminal, digite o comando a seguir para entrar no super usuário do Linux (ter acesso de administrador):

$ sudo su **[ENTER]**

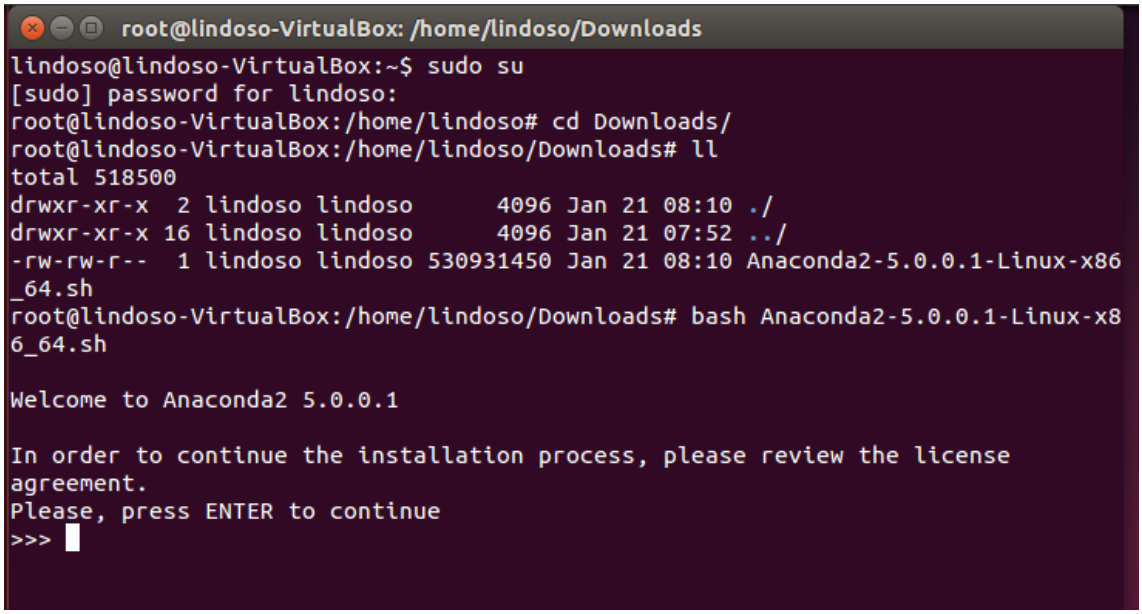
Digite a sua senha.

Agora, localize e instale dentro do terminal o instalador do Anaconda. Por padrão, o navegado Firefox (que já vem instalado com o Ubuntu) armazena o download na pasta /home/seu-usuario/Downloads.

Digite:

$ cd /home/lindoso/Downloads/ **[ENTER]**

$ bash Anaconda2-5.0.0.1-Linux-x86_64.sh **[ENTER]**



Fonte: Autor

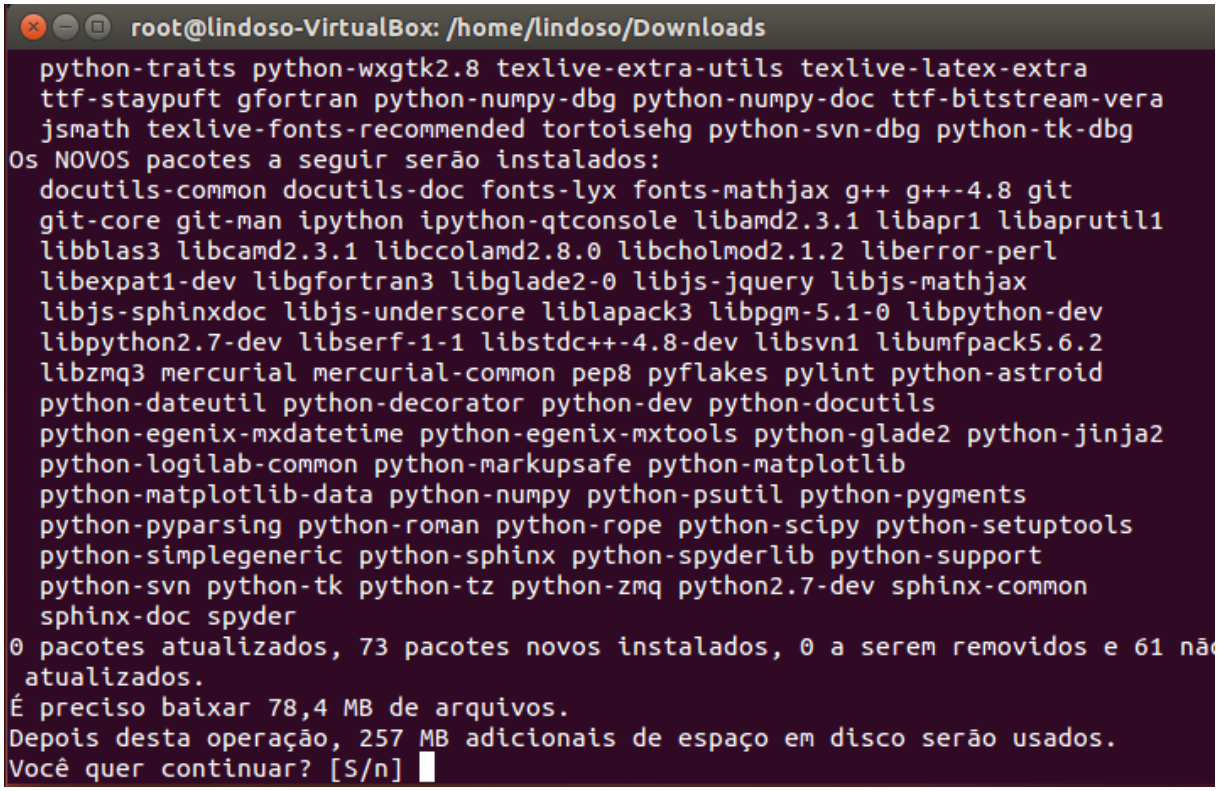
Ao concluir o processo, o Anaconda 2 estará instalado em seu sistema.

1.3.2. Instalando o Spider

No terminal, para instalar o Spyder digite:

$ sudo apt-get update **[ENTER]**
$ sudo apt-get install spyder **[ENTER]**

O download e a instalação demorarão um pouco, pois o Ubuntu terá que instalar todas das dependências que estão faltando em seu sistema. Confirme a operação digitando S.

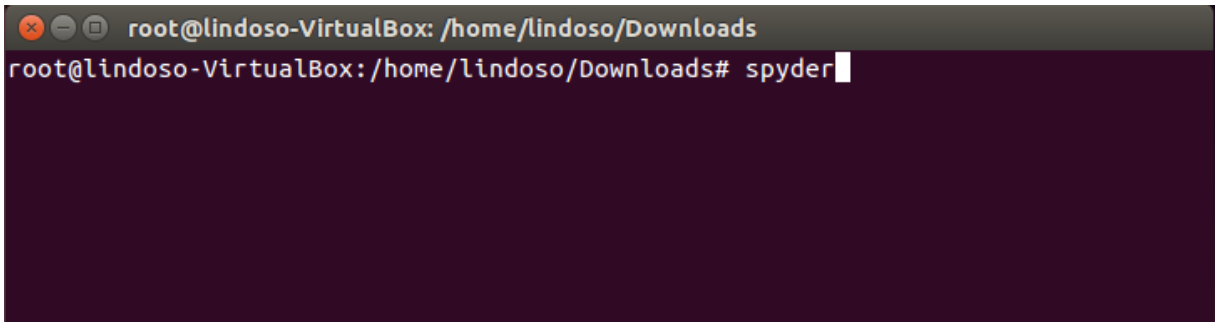


Fonte: Autor

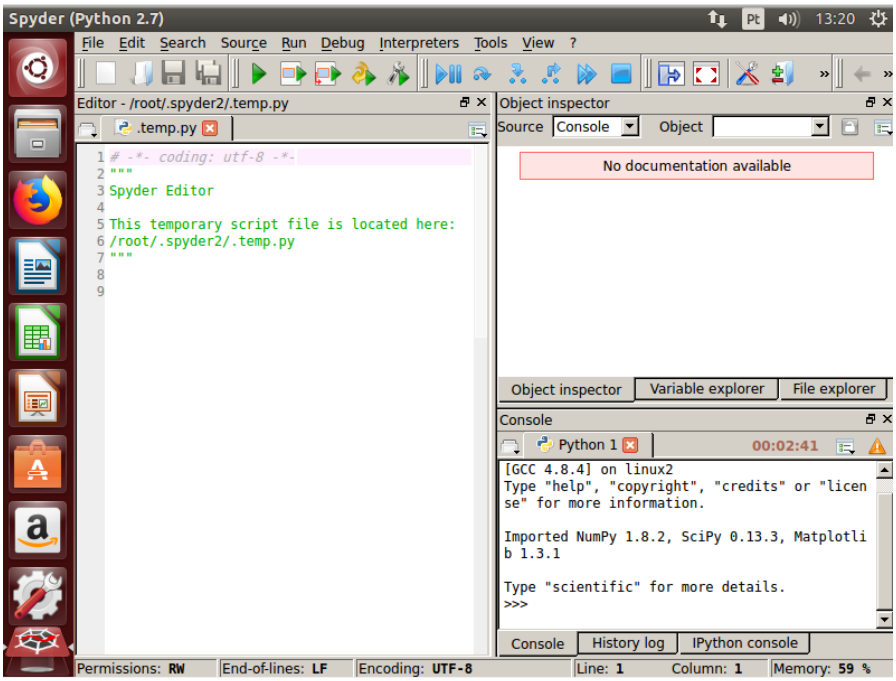
Após finalizado, o Spyder já pode ser utilizado. Para acessá-lo, utilize o comando:

$ sudo su **[ENTER]**

$ spyder **[ENTER]**



Fonte: Autor



Fonte: Autor

1.3.3. Instalando o openopt

No terminal, para instalar o openopt digite:

$ sudo apt-get update **[ENTER]**
$ sudo apt-get install python-openopt **[ENTER]**

Da mesma forma como aconteceu durante a instalação do Spyder, confirme a operação e aguarde sua conclusão.

1.3.4. Instalando o nlopt

No terminal, para instalar o openopt digite:

$ sudo apt-get update **[ENTER]**
$ sudo apt-get install python-nlopt **[ENTER]**

Assim como as duas últimas instalações, confirme a operação e aguarde sua conclusão.

APÊNDICE B – Código: opttruss.py

```
import numpy as np
from openopt import NLP
from matplotlib.pyplot import figure,show

def opttruss(coord,connec,E,F,freenode,V0,plotdisp=False):
    n = connec.shape[0]
    m = coord.shape[0]
    vectors = coord[connec[:,1],:] - coord[connec[:,0],:]
    l = np.sqrt((vectors**2).sum(axis=1))
    e = vectors.T/l
    B = (e[np.newaxis] * e[:,np.newaxis]).T


    def fobj(x):
        D = E * x/l
        kx = e * D
        K = np.zeros((2*m, 2*m))
        for i in range(n):
            aux = 2*connec[i,:]
            index = np.r_[aux[0]:aux[0]+2, aux[1]:aux[1]+2]
            k0 = np.concatenate((np.concatenate((B[i],-B[i]),axis=1), \
                                 np.concatenate((-B[i],B[i]),axis=1)),axis=0)
            K[np.ix_(index,index)] = K[np.ix_(index,index)] + D[i] * k0

        block = freenode.flatten().nonzero()[0]
        matrix = K[np.ix_(block,block)]
        rhs = F.flatten()[block]
        solution = np.linalg.solve(matrix,rhs)
        u = freenode.astype('float').flatten()
        u[block] = solution
        U = u.reshape(m,2)
```

```
        axial = ((U[connec[:,1],:] - U[connec[:,0],:]) * kx.T).sum(axis=1)
        stress = axial / x
        cost = (U * F).sum()
        dcost = -stress**2 / E * l
        return cost, dcost, U, stress

    def volume(x):
        return (x * l).sum(), l

    def drawtruss(x,factor=3, wdt=5e2):
        U, stress = fobj(x)[2:]
        newcoor = coord + factor*U
        if plotdisp:
            fig = figure(figsize=(12,6))
            ax = fig.add_subplot(121)
            bx = fig.add_subplot(122)
        else:
            fig = figure()
            ax = fig.add_subplot(111)
        for i in range(n):
            bar1 = np.concatenate(
(coord[connec[i,0],:][np.newaxis],coord[connec[i,1],:][np.newaxis]),axis=0 )
            bar2 = np.concatenate(
(newcoor[connec[i,0],:][np.newaxis],newcoor[connec[i,1],:][np.newaxis]),axis=0 )
            if stress[i] > 0:
                clr = 'r'
            else:
                clr = 'b'
            ax.plot(bar1[:,0],bar1[:,1], color = clr, linewidth = wdt * x[i])
            ax.axis('equal')
            ax.set_title('Stress')
            if plotdisp:
                bx.plot(bar1[:,0],bar1[:,1], 'r:')
```

```
            bx.plot(bar2[:,0],bar2[:,1], color = 'k', linewidth= wdt* x[i])
            bx.axis('equal')
            bx.set_title('Displacement')
        show()

    xmin = 1e-6 * np.ones(n)
    xmax = 1e-2 * np.ones(n)
    f = lambda x: fobj(x)[0]
    derf = lambda x: fobj(x)[1]
    totalvolume = volume(xmax)[0]
    constr = lambda x: 1./totalvolume * volume(x)[0] - V0
    dconstr= lambda x: 1./totalvolume * volume(x)[1]
    x0 = 1e-4*np.ones(n)
    problem = NLP(f,x0,df=derf,c=constr,dc=dconstr, lb=xmin, ub=xmax,name='Truss',
iprint=100)
    result = problem.solve("mma")

    drawtruss(result.xf)
```

APÊNDICE C – Código: meshtruss.py

```
import numpy as np

def meshtruss (p1,p2,nx,ny):
    nodes = []
    bars = []
    xx = np.linspace(p1[0],p2[0],nx+1)
    yy = np.linspace(p1[1],p2[1],ny+1)
    for y in yy:
        for x in xx:
            nodes.append([x,y])
    for j in range(ny):
        for i in range(nx):
            n1 = i + j* (nx+1)
            n2 = n1 + 1
            n3 = n1 + nx + 1
            n4 = n3 + 1
            bars.append([[n1,n2],[n1,n3],[n1,n4],[n2,n3]])
        index = ny* (nx+1) + 1
        for j in range(nx):
            bars.append([index+j-1,index+j])
        return np.array(nodes) , np.array(bars)
```

APÊNDICE D – Código: exemplo 1.py

```
coord, connec = meshtruss((0,0), (0.6,0.4), 6, 4)
E0=1e+7
E = E0*np.ones(connec.shape[0])
loads = np.zeros_like(coord)
loads[18,1] = -100.
free = np.ones_like(coord).astype('int')
free[::7,:]=0
opttruss(coord,connec,E,loads,free,0.1,True)
```

APÊNDICE E – Código: exemplo 1.1.py (remoção do nó)

```
coord, connec = meshtruss((0,0),(0.6,0.4),6,4)
connec = remove_node(connec,16)
E0=1e+7
E = E0*np.ones(connec.shape[0])
loads = np.zeros_like(coord)
loads[18,1] = -100.
free = np.ones_like(coord).astype('int')
free[16,:]=0
free[::7,:]=0
opttruss(coord,connec,E,loads,free,0.1)
```

APÊNDICE F – Código: examplo1-removingbar.py

```
coord, connec = meshtruss((0,0),(0.6,0.4),6,4)
connec = remove_bar(connec,16,22)
E0=1e+7
E = E0*np.ones(connec.shape[0])
loads = np.zeros_like(coord)
loads[18,1] = -100.
free = np.ones_like(coord).astype("int")
free[::7,:]=0
opttruss(coord,connec,E,loads,free,0.1)
```