UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO MULTIDISCIPLINAR DE CARAÚBAS
CURSO DE ENGENHARIA CIVIL

RENATA DE OLIVEIRA MARINHO

DESENVOLVIMENTO DE UM SOFTWARE PARA DIMENSIONAMENTO DE FUNDAÇÕES DO TIPO TUBULÃO PARA TORRES DE LINHAS DE TRANSMISSÃO

CARAÚBAS - RN

2019

RENATA DE OLIVEIRA MARINHO

DESENVOLVIMENTO DE UM SOFTWARE PARA DIMENSIONAMENTO DE FUNDAÇÕES DO TIPO TUBULÃO PARA TORRES DE LINHAS DE TRANSMISSÃO

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Civil da Universidade Federal Rural do Semi-Árido (UFERSA), Câmpus Caraúbas, como parte dos requisitos para obtenção do Título de Engenheira Civil.

Orientador: Prof. Dr. Manoel Denis Costa Ferreira – UFERSA.

CARAÚBAS – RN

2019

M Marinho, Renata de Oliveira.

337 d DESENVOLVIMENTO DE UM SOFTWARE PARA DIMENSIONAMENTO DE FUNDAÇÕES DO TIPO TUBULÃO PARA TORRES DE LINHAS DE TRANSMISSÃO / Renata de Oliveira Marinho. - 2019.

107 f.: il.

Orientador: Manoel Dênis Costa Ferreira.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia Civil, 2019.

1. Projeto de estruturas. 2. Torres autoportantes. 3. Linhas de transmissão. 4. Python. I. Ferreira, Manoel Dênis Costa , orient.

RENATA DE OLIVEIRA MARINHO

DESENVOLVIMENTO DE UM SOFTWARE PARA DIMENSIONAMENTO DE FUNDAÇÕES DO TIPO TUBULÃO PARA TORRES DE LINHAS DE TRANSMISSÃO

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Civil da Universidade Federal Rural do Semi-Árido (UFERSA), Câmpus Caraúbas, como parte dos requisitos para obtenção do Título de Engenheira Civil.

Defendida em: 22 / 03 / 2019.

BANCA EXAMINADORA

Prof. Dr. Manoel Denis Costa Ferreira (UFERSA)
Presidente

Prof. MSc. Gilvan Bezerra dos Santos Junior (UFERSA)
Membro Examinador

Prof. Aline Ribeiro da Silva (UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço a Deus, por conceder forças e estabilidade para prosseguir o trabalho e por me ajudar a nunca desistir dos meus objetivos.

Ao professor orientador, Manoel Denis, por toda sua ajuda, orientação, esforço e paciência para que o trabalho fosse realizado como planejado.

À minha família que sempre foi e será a base para minhas conquistas, meu pai, Adalcy Marinho, minha mãe, Márcia Verônica, minha irmã, Fernanda Marinho e aos meus avós, Edmilson Oliveira e Ilza Alves que sempre estiverem comigo se esforçando junto a mim para que tudo ocorresse bem e estimulando a alcançar meus planos.

Aos meus irmãos de coração, Alysson Lemos e Fernando Junior, bem como as minhas meninas, Letícia Ainoan, Sara Nunes e Jocélia Dantas que sempre me ajudaram em tudo que precisei, inclusive no decorrer deste trabalho.

A meus amigos, Gleidson leite e João Marcos por todo incentivo e ajuda no presente trabalho.

À toda minha família nascida aqui na UFERSA e na cidade de Caraúbas que sempre estiverem ao meu lado e assim como vários outros que me ajudaram, em especial, Anna Carolina, Weverson Neri, Pedro Praxedes, Marcelo Queiroz, Clark Rasec e Rayla Sousa.

A todos os docentes e discentes que de forma direta ou indireta contribuíram em todo o meu caminhar. De coração, agradeço a todos!

A todos vocês, meu muito obrigado!

Dedico este trabalho com meu imenso amor aos meus pais, Adalcy Marinho e Márcia Verônica, por toda batalha enfrentada para proporcionar uma educação de qualidade, assim como para todos os docentes e discentes que contribuíram para o desenvolvimento deste trabalho.

“*Quando se deseja muito alguma coisa, o universo inteiro conspira para que seu desejo se realize.”*

O Alquimista.

RESUMO

A energia elétrica é um recurso indispensável à vida moderna. Neste sentido, o desenvolvimento de uma região é fortemente impulsionado pelo crescimento do setor elétrico, no qual o Brasil com sua vasta extensão territorial e grande capacidade em recursos energéticos, em sua grande maioria renovável, porém afastadas dos centros consumidores, requer a construção de uma extensa rede de transmissão de energia. Sendo as linhas de transmissão as responsáveis por transmitir energia ou sinais a longas distâncias e tendo em vista sua importância no contexto do setor elétrico do país, é primordial desenvolver um bom projeto destas construções desde sua fundação até os elementos que compõe sua estrutura em si, para que as mesmas resistam aos esforços submetidos e desempenhem seu papel de forma segura. O dimensionamento de fundações de torres de linhas de transmissão de energia elétrica geralmente é desenvolvido a partir das formulações típicas para projetos das construções correntes. O presente trabalho apresenta o desenvolvimento de um software para auxiliar no projeto dos elementos de fundações de torres autoportantes de linhas de transmissão conforme as normas vigentes no país e utilizando a linguagem de programação Python. Para validar os resultados obtidos pelo software implementado apresenta-se um exemplo de aplicação com os resultados comparados ao obtidos de uma solução analítica. Com a análise dos resultados, conclui-se que o software desenvolvido proporciona os resultados esperados, garantindo eficiência no dimensionamento de fundações de torres de linhas de transmissão de energia elétrica.

PALAVRAS-CHAVES: Projeto de estruturas; Torres autoportantes; Linhas de Transmissão; Python.

ABSTRACT

Electricity is an indispensable resource for modern life. In this sense, the development of a region is strongly driven by the growth of the electric sector, in which Brazil, with its vast territorial extension and great capacity in energy resources, largely renewable, but away from the consumer centers, requires the construction of a extensive network of energy transmission. Since transmission lines are responsible for transmitting energy or signals at long distances and in view of their importance in the context of the country's electricity sector, it is essential to develop a good design of these constructions from its foundation to the elements that make up its structure, so that they resist the submitted efforts and play their role in a safe way. The design of tower foundations of electric power transmission lines is usually developed from the typical formulations for projects of the current constructions. The present work presents the development of a software to assist in the design of the foundations elements of freestanding towers of transmission lines according to the norms in force in the country and using the Python programming language. In order to validate the results obtained by the implemented software, an example of application with the results compared to that obtained from an analytical solution is presented. With the analysis of the results, it is concluded that the developed software provides the expected results, guaranteeing efficiency in the design of foundations of towers of electric power transmission lines.

KEYWORDS: Project in structure; Self-supporting towers; Transmission lines; Python.

SUMÁRIO

LISTA DE ANEXOS

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

Devido ao grande crescimento urbano, há a necessidade da construção de grandes centrais de produção de energia elétrica para suprir a necessidade de tal desenvolvimento, onde se torna fundamental o uso de uma extensa rede de transmissão de energia. A construção de linhas de transporte de energia depende de fatores, como a topografia do terreno, a composição geológica do mesmo afetando o tipo de fundação que será utilizada nas superestruturas (AFONSO,2015).

As linhas de transmissão são compostas por torres metálicas treliçadas, cabos, para-raios e isoladores que trilham longos percursos. No decorrer dos trajetos, as torres são implantadas em solos de origem e propriedades geotécnicas variadas, como já mencionado. E tendo em vista sua importância no contexto do setor elétrico do país, é primordial desenvolver um bom projeto destas construções desde sua fundação até os elementos que compõe sua estrutura em si, para que as mesmas resistam aos esforços submetidos e desempenhe seu papel de forma segura (AMARAL,2015).

Até meados do século XIX, os projetos estruturais eram desenvolvidos manualmente e impossibilitava análises mais realistas dos carregamentos e comportamentos das torres. Por volta de 1970, iniciou-se a popularização dos computadores e foi introduzido o cálculo eletrônico na engenharia estrutural, possibilitando as mais variadas combinações na relação altura/carregamentos. Com isso, as reações nas fundações ficaram mais reais e seus valores mais concretos (CHAVES,2004).

Nesse contexto, o presente trabalho apresenta o desenvolvimento de um software para auxiliar no projeto do elemento de fundação do tipo tubulão para torres autoportantes de linhas de transmissão conforme as normas vigentes no país e utilizando a linguagem de programação Python.

1.1 Objetivos gerais

Este trabalho tem como objetivo geral desenvolver um programa para o dimensionamento de fundação do tipo tubulão direcionada a torres de linhas de transmissão.

1.2 Objetivos específicos

- Determinação dos parâmetros para o dimensionamento da fundação do tipo tubulão direcionada a torres de linhas de transmissão;
- Desenvolver os códigos na linguagem Python, para o dimensionamento de tubulões;
- Validar o programa desenvolvido por meio de exemplificações.

2 REFERENCIAL TEÓRICO

2.1 Torres de linhas de transmissão

As torres de linhas de transmissão são elementos estruturais responsáveis por sustentar cabos e acessórios afim de transmitir energia elétrica entre subestações (QUENTAL, 2008). Tais elementos estruturais são, usualmente, compostos por perfis de aço galvanizado, no qual devem respeitar limites de segurança, desempenho e custo (VELOZO, 2010).

As torres de linhas de transmissão e telecomunicação comumente usadas podem ser denominadas e agrupadas de diversas formas, no qual sua funcionalidade estrutural (função na linha) e seu sistema estrutural são os mais importantes para este estudo (VELOZO, 2010).

2.1.1 Características quanto à sua função na linha

As torres são divididas, simplificadamente, em três categorias básicas: terminais ou fim de linha, torres de suspensão e torres em ângulo (CHAVES, 2004).

As torres definidas como terminais ou fim de linha, como o próprio nome sugere, são dispostas no início ou fim do traçado das linhas de transmissão, com a finalidade de ancorar os esforços oriundos dos cabos condutores e dos cabos pára-raios. Essas torres possuem estruturas robustas e resistem a esforços dos cabos em ângulos, no qual o seu eixo não coincide com o eixo da linha (VELOZO, 2010).

As torres de suspensão são aquelas locadas em locais de trecho reto, ou que possuem pouca angulação, ângulos menores que cinco graus (CHAVES, 2004).

As torres em ângulo são adequadas em locais onde necessita de mudança de direção do traçado da linha. Tais torres, também são denominadas como torres de ancoragem, pois resistem às resultantes dos esforços nas diagonais das direções entre seus eixos (VELOZO, 2010).

2.1.2 Características quanto ao seu sistema estrutural

São subdivididas em dois grupos: autoportantes e estaiadas.

As torres autoportantes são aquelas que mantém seu equilíbrio em função da sua estrutura, sem que seja necessária uma outra estrutura de suporte, como representado na figura 1. Esse tipo de torre é a mais usual em toda a extensão do território brasileiro (CHAVES, 2004).

As torres estaiadas são estruturas que utilizam estais, no qual são elementos que reagem à tração em função dos carregamentos horizontais, especialmente, as ações do vento (AMARAL, 2015). Esses estais são fixados na parte superior da estrutura e possuem um espraiamento de 30º com a vertical, conforme a figura 2. A escolha de uso de uma estrutura autoportante ou estaiada é em função da topografia de instalação, onde as autoportantes são adequadas para terrenos mais acidentados devido a sua forma mais compacta e as estruturas estaiadas podem ser aplicadas em terrenos mais suaves com espaço para ancoragem dos estais. A vantagem do uso de torres estaiadas é devido serem mais leves, tornando-se menos onerosas que as autoportantes (AMARAL, 2015).

Segundo Chaves (2004), as torres autoportantes são utilizadas em terminais ou fim de linha, bem como são usuais em mudanças de direção. Ao passo que, para o uso em suspensão é indicada as torres estaiadas.

Figura 1 – Silhueta de Torre Autoportante.

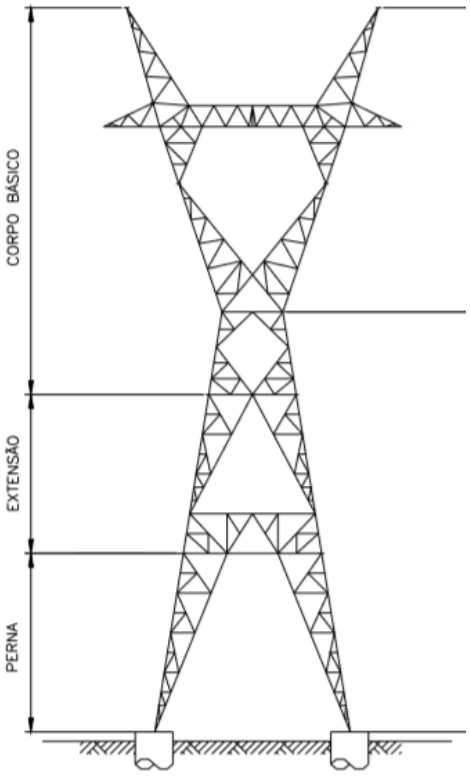
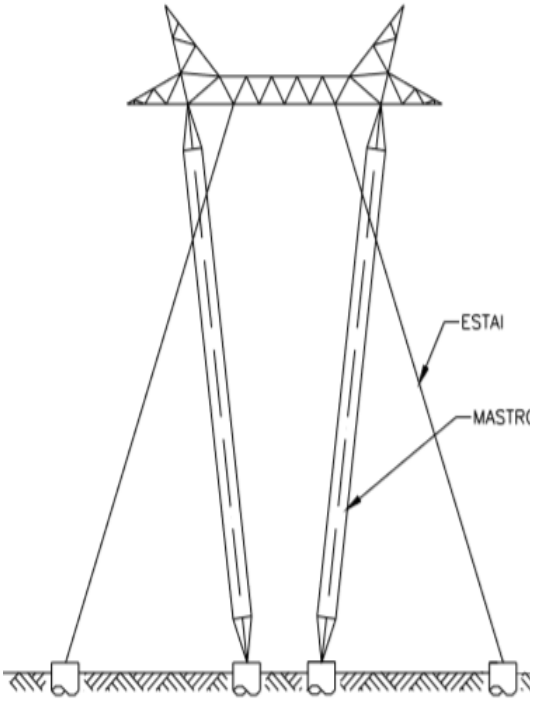


Figura 2 – Silhueta de Torre Estaiada.



Fonte: CHAVES, 2004.

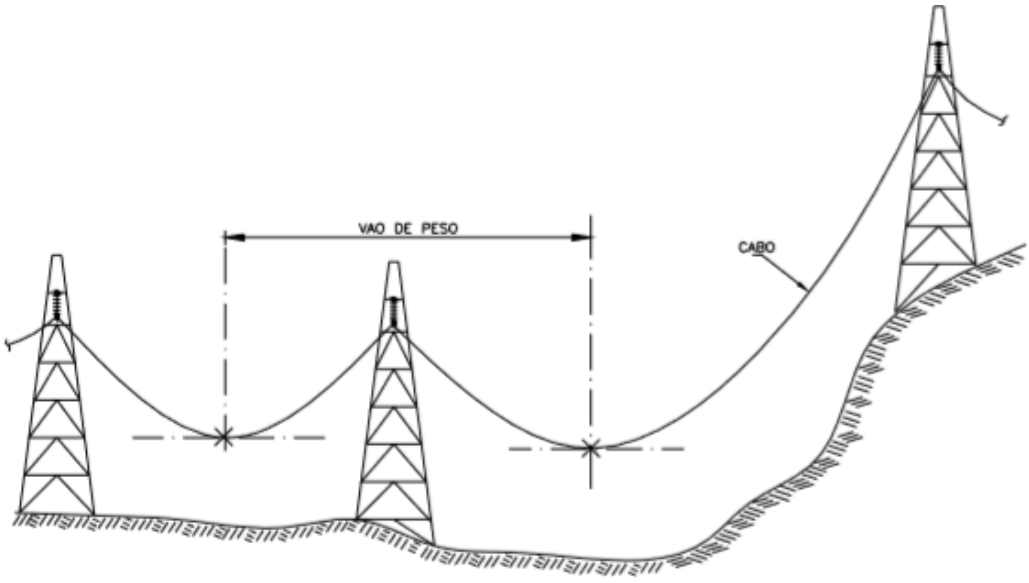
2.2 CARREGAMENTOS NAS TORRES

Os esforços atuantes na fundação são devidos às ações em que a torre está sujeita. Tais esforços possuem formas distintas de atuação vetorial, podendo ser através de cargas verticais, longitudinais e transversais (AMARAL, 2015).

As cargas verticais exercidas nas torres são provenientes da força gravitacional dos seus elementos constituintes: peso próprio da torre, da cadeia de isoladores e do peso que advém de cabos condutores e pára-raios. Para o peso relacionado aos cabos da torre refere-se ao vão gravante ou vão de peso, o qual este vão é a distância horizontal entre pontos que têm tangente horizontal

com as catenárias dos vãos adjacentes à torre, conforme ilustrado na figura 3 (CHAVES, 2004).

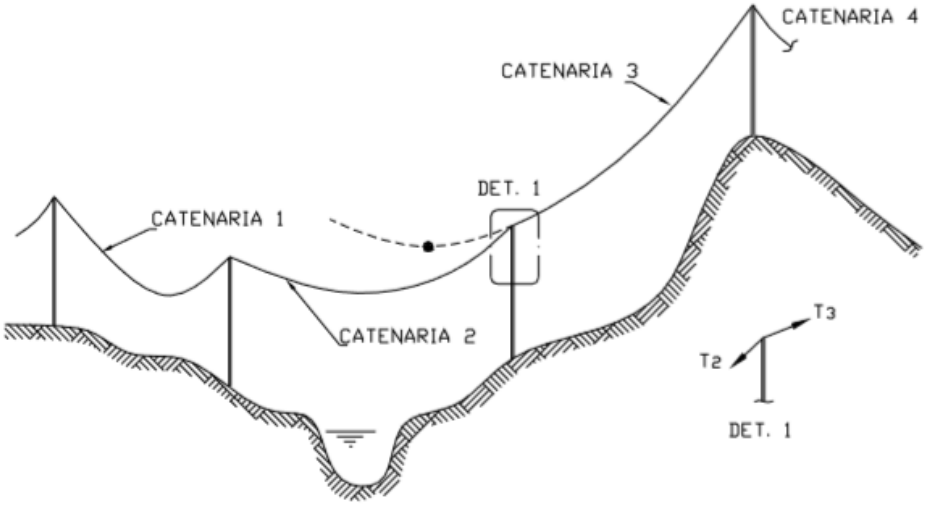
Figura 3 – Vão gravante de uma torre.



Fonte: CHAVES, 2004.

Essas cargas verticais exercidas nas torres, nem sempre atuam de cima para baixo. Há situações nas quais o cabo introduz um arrancamento na torre, como pode se observar na situação da figura 4. Quando isto acontece, diz-se que a torre está sofrendo uma “carga reduzida” e isto deve ser previsto no dimensionamento (CHAVES, 2004).

Figura 4 – Torre sob arrancamento parcial.



Fonte: CHAVES,2004.

A NBR 5422:1985 recomenda ponderar as cargas permanentes com fatores mínimos de K_1 e K_2, onde:

K_1 – 1,15 para cargas máximas de peso de cabos;

K_2 – 1 para o peso próprio da torre, para as ferragens dos cabos, para as cadeias de isoladores e para cargas verticais reduzidas.

Pode-se calcular o valor da carga vertical da torre provinda do peso dos cabos, segundo a equação (1).

$$\mathbf{V_C = K_1.P_c.}V_g \quad (1)$$

Onde:

V_g – é o vão gravante ou vão de peso;

P_c – é o peso do cabo por unidade de comprimeto;

K_1 – é tomado igual a 1,15 no caso de vão gravante máximo. E será tomado como igual a 1,00 para vão gravante mínimo, ou no caso de vão reduzido.

O peso da cadeia e das suas ferragens é calculado a partir da equação (2).

$$\mathbf{V_I = K_2.P_{cf}} \quad (2)$$

Onde:

P_{cf} – é o peso da cadeia de isoladores;

K_2 – é tomado igual a 1.

Há também as cargas longitudinais resultantes da tração que os cabos de circuito e para-raios estão sujeitos. Por fim tem-se as cargas transversais que ocorrem devido à ação do vento nos elementos das torres (AMARAL, 2015).

As estruturas também terão de contemplar cargas periódicas ou acidentais que irão ocorrer ao longo da vida útil da estrutura, como por exemplo, manutenção, montagem da estrutura da torre e possíveis rompimentos de cabos, dentre outros (AMARAL, 2015).

Contudo, a avaliação numérica da ação do vento sobre as linhas de transmissão é regida de acordo com a norma brasileira NBR 5422:1985, onde também está relatado todos os procedimentos para quantificação das cargas

atuantes nas torres. A NBR 5422 refere-se à ação do vento se embasando em uma pressão dinâmica de referência q_0 calculada a partir da equação (3).

$$q_0 = \frac{1}{2}\rho V_p^2 \qquad (N/m^2) \qquad (3)$$

Onde:

ρ – Massa específica do ar, em kg/m³;

V_p – velocidade do vento de projeto, em m/s.

Segundo Chaves (2004), as hipóteses de carregamento tentam mostrar as prováveis circunstâncias em que as torres poderão ser submetidas. Essas torres, normalmente, apresentam um número de hipóteses de carregamento muito grande e tais hipóteses são demonstradas em “árvores” de carregamento, onde as mesmas servem de referência para a montagem das cargas nas torres.

São apresentadas sugestões e condições, por alguns autores, para essas hipóteses de carregamento e estão dispostas a seguir (GONTIJO, 1994):

- Hipótese 1 – vento máximo em qualquer direção, e cabos sem romper. Usualmente, aplica-se ventos a 0, 45 e 90 graus em relação ao eixo da torre;

- Hipótese 2 – vento com velocidade reduzida, com um cabo para-raios rompido;

- Hipótese 3 – vento com velocidade reduzida, e um cabo condutor rompido;

- Hipótese 4 – cargas devidas à construção, ou montagem, com o lançamento dos cabos condutores e para-raios.

Conforme Chaves (2004), todos os possíveis aspectos (montagem) da torre, já estão inseridos nas hipóteses de carregamento; isso relativo as combinações de extensão e de alturas de pés.

Analisando as principais hipóteses críticas na qual as fundações são submetidas, pode-se citar as seguintes (CHAVES, 2004):

- Compressão máxima com horizontais correspondentes;

- Tração máxima com horizontais correspondentes;

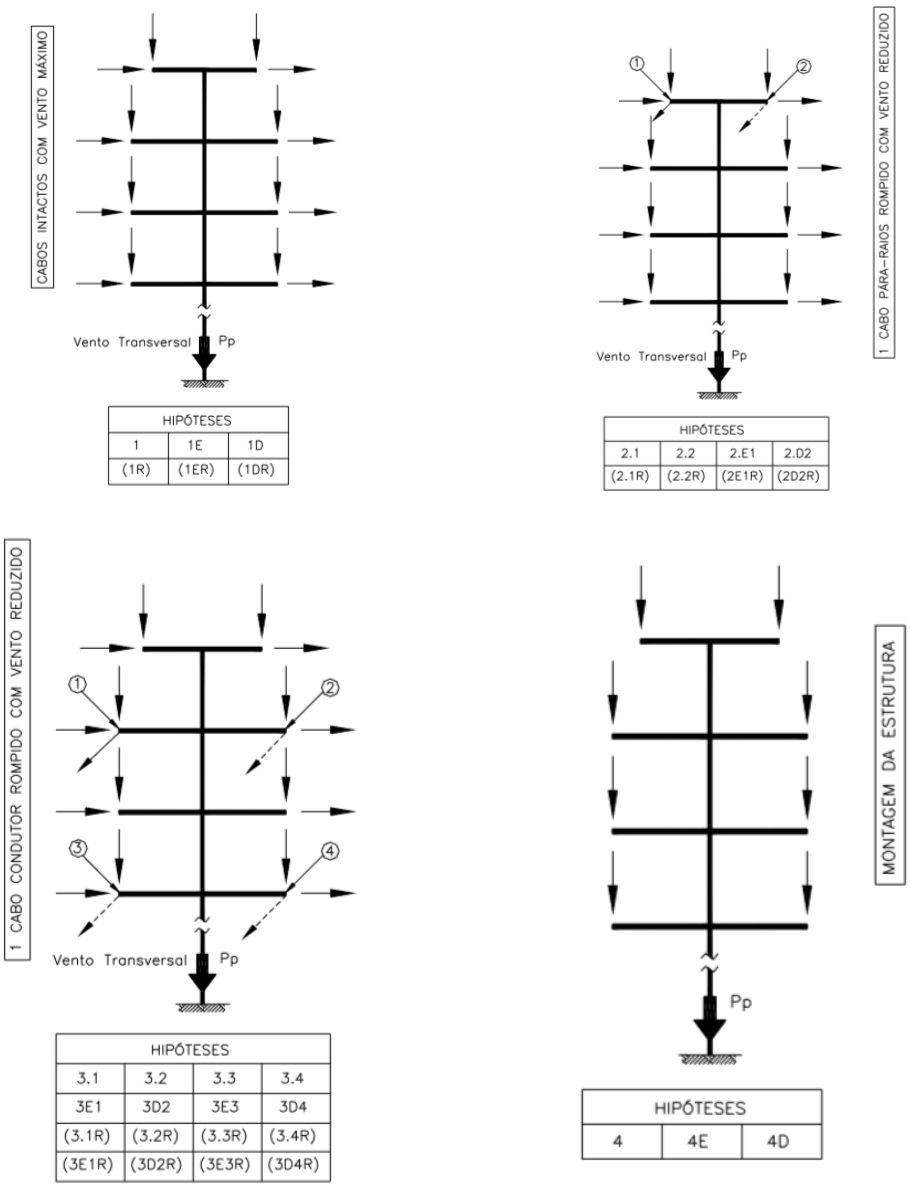
- Força horizontal transversal máxima;

- Força horizontal longitudinal máxima.

Chaves (2004), ressalta que o fato de as torres apresentarem estruturas treliçadas, junto a nós articulados, considera-se que elas não aplicam momentos fletores nas fundações em seus pontos de apoio.

As ações nas quais as torres estão sendo submetidas são reunidas e apresentadas em desenhos esquemáticos de “árvores” de carregamentos, como apresentado na figura 5.

Figura 5: Árvore de carregamento em torres - Hipótese básica.



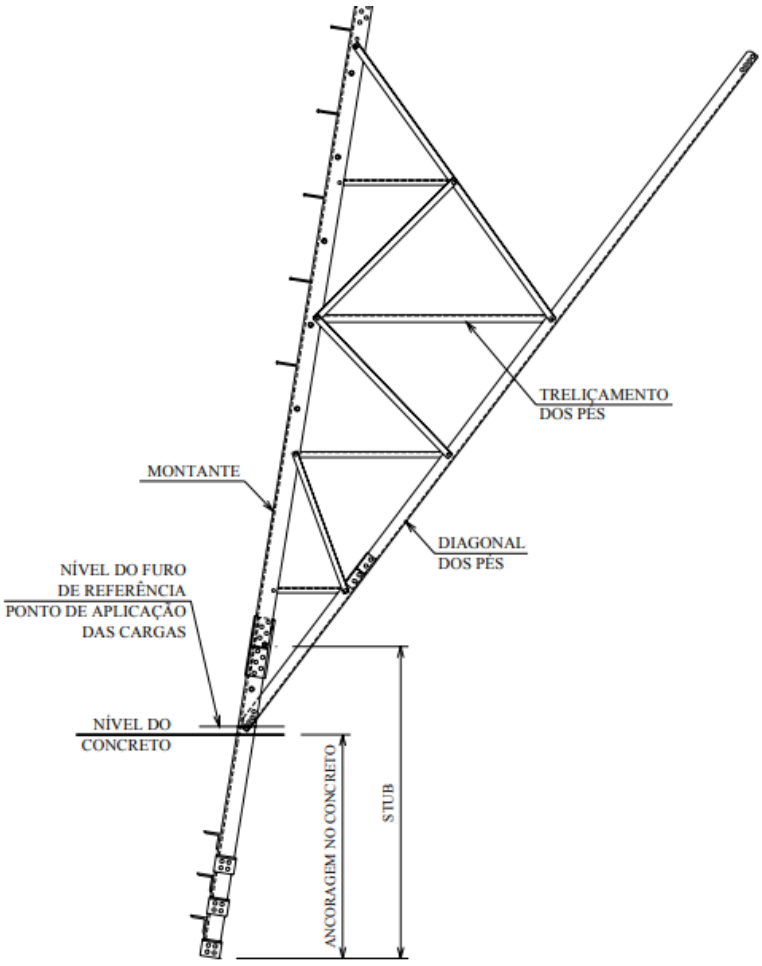
Fonte: CHAVES, 2004.

2.3 Ligação entre a torre e a fundação

As estruturas de fundações na construção civil, em geral, possuem uma armadura de arranque para promover a ligação entre a fundação e a estrutura e, consequentemente, a continuação da distribuição de esforços (AMARAL, 2015).

Entretanto, as fundações das torres de linhas de transmissão não possuem esse tipo de armadura e a ligação da superestrutura é feita através de uma cantoneira metálica munida de aletas que irá promover uma ancoragem eficiente na interface do concreto, no qual esta peça é denominada de stub. A figura 6 representa a locação do stub, no qual é pouco provável que tenha forma diferente, ou seja, formato retilíneo com inclinação igual aos montantes da estrutura (VELOZO, 2010).

Figura 6 – Ligação entre a torre autoportante e a fundação em concreto.



Fonte: GARCIA, 2005.

2.4 Tipos de fundações usadas nas torres de linhas de transmissão

As fundações utilizadas em linhas de transmissão podem ser do tipo rasas e profundas. As fundações rasas são definidas como aquelas em que se assentam em pequenas profundidades, pois seu solo apresenta resistência suficiente, em cotas pouco profundas, compatíveis com o projeto. Já as fundações profundas são aquelas que alcançam a resistência compatível com o projeto em profundidades bem maiores (VELLOSO E LOPES, 2010).

A NBR 6122:1996 determina como limite entre esses grupos que a razão de profundidade de assentamento com a menor dimensão da base da fundação, seja maior duas vezes para considerá-la profunda. As fundações mais usuais em linhas de transmissão são os tubulões e as sapatas, porém neste trabalho será limitado aos tubulões (fundação profunda). Já para fundações rasas, é corriqueiro o uso de sapatas.

2.4.1 Tubulões

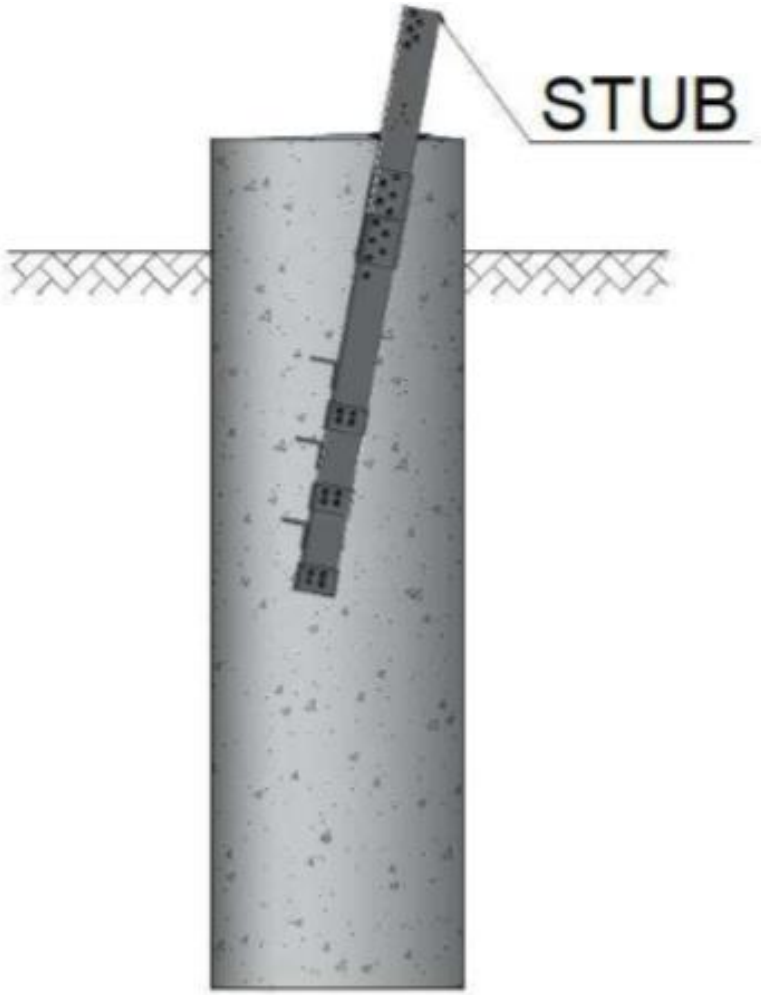
A determinação do uso de tubulões se dá quando não se encontra uma capacidade de suporte do solo mais a superfície e o assentamento desta fundação será feito em camadas mais subjacentes, procurando obter uma capacidade de suporte adequada, ou seja, depende da carga a ser transmitida e da resistência do solo (REBELLO, 2008).

Este elemento de fundação é composto por um cilindro vertical, denominado de fuste e pode ter sua base alargada ou não, onde sua base pode ser circular ou em forma de elipse, no qual a NBR 6122:1996 determina que a altura desta base não pode ultrapassar 2 m e a largura do seu fuste deve ter no mínimo 70 cm.

Segundo Chaves (2004) as fundações do tipo tubulões são usuais devido ao seu baixo custo e isto, provém do seu volume de escavação que é, relativamente, pequeno. O tubulão não necessita de reaterro, exige um pequeno consumo de formas, sua execução provoca pouca interferência no local a ser implantado, reduzindo e/ou eliminando a necessidade de recomposição vegetal.

Pode-se citar também como vantagem, que sua execução não provoca impactos e despreza a necessidade de medidas de proteção a terreno ou construções vizinhas (REBELLO, 2008). Outra vantagem é a proteção contra ações ambientais que esse tipo de fundação proporciona para a cantoneira de ancoragem, o stub. A figura 7 mostra uma representação desse esquemático.

Figura 7 – Esquemático de tubulão com stub.



Fonte: Amaral,2015.

Em relação a sua geometria, Rebello (2008) relata que a sua base alargada se comporta como uma sapata isolada, sendo submetida a flexão e obtendo forças de tração na sua face inferior. De acordo com Chaves (2004) as bases alargadas permitem que a carga de compressão seja propagada no solo com tensões baixas e compatível com tensão admissível do solo naquela cota.

Os tubulões possuem geometria simples, já que a mesma é alinhada pela segurança da escavação e por isso, possui seção circular. No caso das linhas

de transmissão, os fustes possuem diâmetros entre 60 a 100 cm. Quando os tubulões são instalados em terrenos propensos ao desmoronamento, geralmente, coloca-se um bloco de coroamento em seu topo. Se não houver problemas com desmoronamento o tubulão aflora com sua própria seção circular (CHAVES, 2004).

2.5 Resistência do solo quanto às solicitações das estruturas de fundações

No dimensionamento de fundações para torres de linhas de transmissão, considera-se esforços de compressão, arrancamento e tombamento (QUENTAL, 2008).

2.5.1 Resistência à compressão

Segundo Chaves 2004, para o estudo à compressão determina-se a cota de assentamento e a tensão admissível do solo de acordo com o relatório de sondagem.

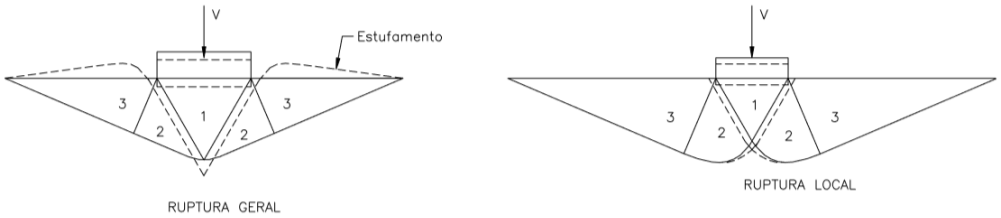
A carga admissível também pode ser calculada a partir dos métodos respaldados na mecânica dos solos, no qual segue o modelo idealizado por Terzaghi para ruptura do solo (AMARAL, 2015).

Chaves 2004, retrata sobre o modelo de Terzaghi afirmando que:

> Sua proposição sugere a formação de uma cunha sob a fundação. O movimento vertical desta porção do solo, devido ao carregamento, mobiliza o solo adjacente estimulando o aparecimento de duas zonas de cisalhamento: uma de cisalhamento radial (admitida ser limitada por um arco de espiral logarítmica), e por outra de cisalhamento linear.

A figura 8 apresenta as zonas de cisalhamento radial e linear, respectivamente.

Figura 8 – Mecanismos de ruptura.



Fonte: Chaves, 2004.

Seguindo esse modelo obtém-se uma equação que resulta na tensão de ruptura do solo (q_u) e com um fator de segurança determina-se a tensão admissível (q_s). Quando se faz uso de fórmulas empíricas, geralmente, utiliza-se um fator de segurança igual a 3 (CHAVES, 2004).

$$q_{adm} = {}^{q_u}/_{FS} \quad (4)$$

A ruptura do solo pode acontecer de maneira geral ou local, como Terzaghi determinou inicialmente, no qual para ele solos mais compactos e rijos possuem ruptura do tipo geral. As rupturas locais ocorrem em solos mais fofos e Terzaghi indicou fatores de minoração para a capacidade de carga e redução do valor da coesão (VELLOSO e LOPES, 2010). Terzaghi propõe uma equação para determinação da tensão de ruptura do solo, na qual considera-se a forma geométrica da fundação:

$$q_u = cN_CS_C + 0{,}5\gamma BN_\gamma S_\gamma + qN_qS_q \quad (5)$$

Onde:

qu – é a tensão de ruptura do solo (tensão última);

c – é a coesão do solo;

Nc, Nγ e Nq – são fatores de capacidade de carga do solo (função do ângulo de atrito φ);

q – é o valor da tensão efetiva do solo na cota de assentamento da fundação. No caso de se considerar apenas o embutimento da fundação esse valor é tomado como o peso do solo acima daquela profundidade h, ou seja, q = γh;

B – é a menor dimensão da fundação (L é a maior);

Sc, Sγ e Sq – são fatores de forma (função da forma da fundação).

Para a ruptura do tipo local utiliza-se a mesma equação (5), entretanto com redução dos valores de coesão e ângulo de atrito, como mostrado na nas equações (6) e (7):

$$c' = \frac{2}{3} c \quad (6)$$

$$\varphi' = \tan^{-1}\left(\frac{2}{3}\tan\varphi\right) \quad (7)$$

Segundo Cintra, Aoki e Albiero (2011) os fatores de capacidade de carga dependem apenas do ângulo de atrito do solo e são determinados segundo as equações (8), (9) e (10).

$$N_q = \tan^2(45+0{,}5\varphi)e^{\pi\ tang\varphi} \quad (8)$$

$$N_c = \cot\varphi\ (N_q - 1) \quad (9)$$

$$N_\gamma \cong 2tang\ \varphi\ (N_q + 1) \quad (10)$$

Já os fatores de forma para fundações com base quadrangular e circular são calculadas com as equações (11), (12) e (13).

$$S_q = 1 + \tan\varphi \quad (11)$$

$$S_c = 1 + \frac{N_q}{N_c} \quad (12)$$

$$S_\gamma = 0{,}60 \quad (13)$$

A tensão admissível também pode ser determinada através de correlações do seu ensaio de SPT, onde é feita sua associação empírica com o número de golpes N obtidos no processo de sondagem à percussão (CHAVES, 2004). Entretanto, esse processo tem várias imposições que devem ser seguidas à risca.

No estudo a compressão, considera-se a contribuição do atrito lateral, entretanto é desprezado o trecho de comprimento igual ao diâmetro da base alargada, abrangendo a partir do trecho que se inicia o alargamento (figura 9), de acordo com a NBR 6122.

Segundo CESP (1983) *apud* Chaves (2004) a resistência lateral é analisada por meio da tensão de atrito, na qual é calculada através do número de golpes do SPT, conforme os valores a seguir:

- Solo seco:

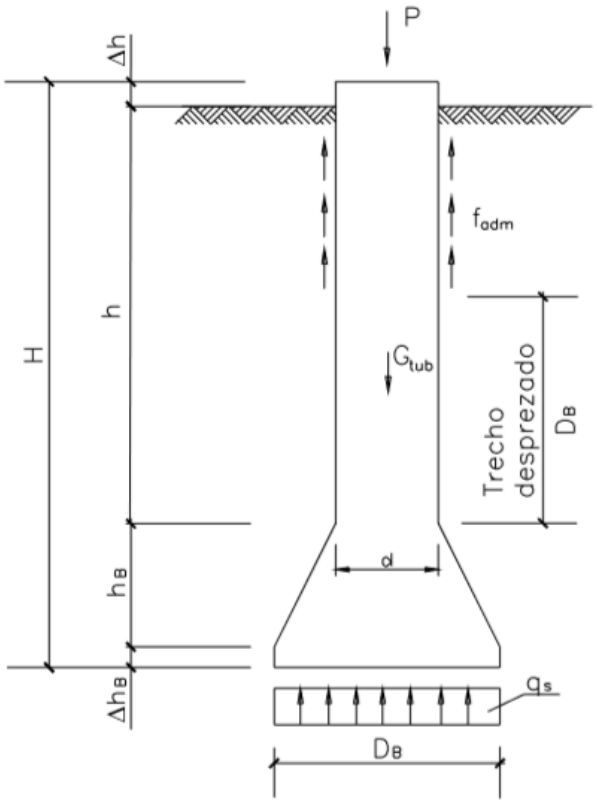
  - $2 \leq N \geq 4 \rightarrow f_{adm} = 10 \frac{kN}{m^2}$
  - $4 \leq N \geq 8 \rightarrow f_{adm} = 16 \frac{kN}{m^2}$
  - $8 \leq N \geq 20 \rightarrow f_{adm} = 16 \text{ a } 30 \frac{kN}{m^2}$

- Solo saturado:

  - $f_{adm} = 10 \frac{kN}{m^2}$

Tais valores apresentados para f_{adm} são tensões admissíveis.

Figura 9 – Esquema do equilíbrio vertical dos tubulões.



Fonte: Chaves,2004.

Com isso, calcula-se a tensão atuante na base a partir da equação (14).

$$Q_s = \frac{P + G_{tub} - A_l f_{adm}}{Ab} \quad (14)$$

onde:

P – é a carga de compressão vertical aplicada no tubulão;

Gtub – é o peso próprio do tubulão;

Al – é a área lateral onde atua o atrito – **Al = πd (h – Db);**

fadm – é a tensão de atrito na superfície lateral;

Ab – é a área da base – **Ab = πDb² / 4.**

2.5.2 Resistência ao arrancamento (Tração)

Segundo Chaves (2004), existe vários métodos que analisam a carga última de arrancamento, na qual uma das mais difundidas é a metodologia do cone invertido.

Danziger (1983), relata que a esse método estabelece que a capacidade de carga de uma fundação, quando submetida ao esforço de tração é o peso próprio da fundação somado ao peso da terra contido num tronco de cone invertido, onde a base menor corresponde a base da fundação, a geratriz formando um ângulo α com a vertical, e a base maior consiste na interseção da superfície lateral com o terreno, conforme está apresentado na figura 10.

Esse ângulo de arrancamento do solo é adotado com 20º para areias e 30º para argilas, conforme o *Department of the Navy – Naval Facilities Engineering Command – Design Manual – Soil mechanics, foundations, and earth structures – Navfac dm-7.*

Com isso, o equilíbrio vertical é expresso por:

$$P_u = G_{tub} + P_s \quad (15)$$

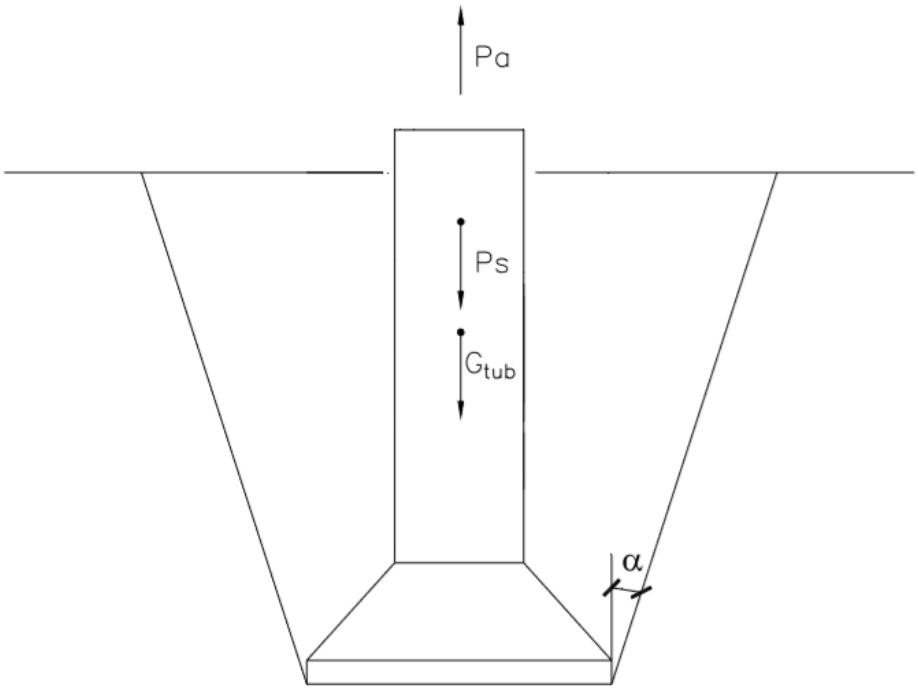
Onde:

Pu – é a carga máxima que pode ser mobilizada – carga última;

Gtub – é o peso próprio do tubulão;

Ps – é o peso do volume de solo do tronco de cone.

Figura 10 – Esquematização do método do cone invertido com o equilíbrio vertical à tração.



Fonte: Chaves, 2004.

2.6 Linguagem de programação python

2.6.1 Conceitos básicos

De acordo com Borges (2010), Python é uma linguagem de programação orientada a objeto, aliada a uma tipagem dinâmica, forte e interativa. O Python dispõe de uma sintaxe clara e concisa, favorecendo a legibilidade e a tornando mais produtiva. A linguagem possui estruturas de alto nível e uma grande coleção de módulos.

O Python também é aplicado em funções de *script* e possuem um propósito geral e, normalmente, as pessoas usam o termo *script* ao invés de “programa” para retratar a um arquivo de código de Python (LUTZ e ASCHER, 2007).

Dentre os inúmeros atributos da linguagem que a tornam relevante para a computação científica, Coelho (2007) destaca: a multiplataforma, portabilidade, software livre, extensibilidade, orientação a objeto, tipagem automática, tipagem forte, código legível, flexibilidade, operação com arquivos e uso interativo. A orientação a objetos é uma característica chave dessa linguagem, já que tudo em Python é um objeto, ou seja, funções, variáveis de todos os tipos e até mesmo módulos são objetos.

2.6.2 Interpretador Python

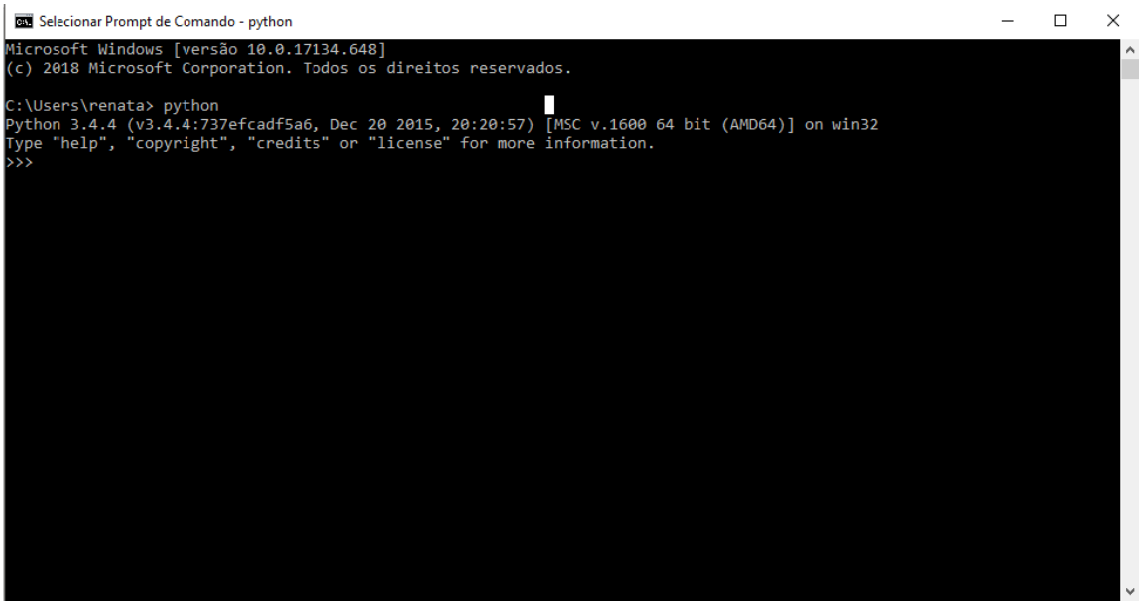
Interpretador é um programa de computador que executa orientações escritas em uma linguagem de programação. Um interpretador faz uso de estratégias para execução do programa, onde o mesmo executa o código fonte diretamente ou traduz o mesmo em alguma representação intermediária para, por fim, executá-lo (SOUSA FILHO e ALEXANDRE, 2015).

A linguagem de programação Python possui uma forma interativa de interpretador, onde as linhas de código são expressas em um *prompt* (linha de comando) e esse modo interativo é o diferencial do Python (BORGES, 2010). O código fonte dessa linguagem é traduzido para *bytecode*, no qual representa um formato binário com instruções para o interpretador. A figura 11 apresenta o layout de um interpretador em Python.

Borges (2010), explica o uso do *bytecode* dentro do interpretador:

> Por padrão, o interpretador compila o código e armazena o *bytecode* em disco, para que a próxima vez que o executar, não precise compilar novamente o programa, reduzindo o tempo de carga na execução. [...] Quando um programa ou um módulo é evocado, o interpretador realiza a análise do código, converte para símbolos, compila (se não houver *bytecode* atualizado em disco) e executa na máquina virtual Python. O *bytecode* é armazenado em arquivos com extensão “.pyc” (*bytecode* normal) ou “. pyo” (*bytecode* otimizado). O *bytecode* também pode ser empacotado junto com o interpretador em um executável, para facilitar a distribuição da aplicação, eliminando a necessidade de instalar Python em cada computador.

Figura 11 – Layout de um interpretador Python.



Fonte: Autoria própria.

2.6.3 Funções

Conforme Borges (2010) as funções podem ser entendidas como blocos de código indicadas por um nome, que possibilitam receber parâmetros pré-determinados. No Python, as funções possuem características como: podem retornar ou não objetos, aceitam *Doc Strings*, aceitam parâmetros opcionais, aceitam que os parâmetros sejam passados com nome, dentre outros. A figura 12 representa uma função executável no Python.

Figura 12 – Representação de uma função.

```
def func(parametro1, parametro2=padrao):
    """Doc String
    """
    <bloco de código>
    return valor
```

Fonte: Borges, 2010.

2.6.4 Orientação a objetos

A programação orientada a objetos surgiu da necessidade de simular a realidade através de algo abstrato do cotidiano, afirma Chagas e Oliveira (2009), onde o seu conceito básico é retratar as características pertinentes dos objetos envolvidos no sistema computacional em desenvolvimento.

Essa simulação da realidade, ocorre de forma diferente da Programação Orientada a Procedimentos (POP), já que se incluiu a concepção de visibilidade na orientação a objetos. A visibilidade é dita como o nível de acesso de propriedades e métodos de uma classe, essa definição limita o acesso aos métodos, impedindo o acesso direto aos atributos, tornando possível controlar as operações (CHAGAS e OLIVEIRA, 2009).

Com a finalidade de tornar esse paradigma um excelente meio de desenvolvimento de software, seus criadores e idealizadores produziram diversas definições que estão listadas na Tabela 1.

Tabela 1 – Principais definições sobre a programação orientada a objetos.

Classe	Representa um conjunto de objetos com características comuns. Uma classe define o comportamento dos objetos, através de métodos, e quais estados ele é capaz de manter, através de atributos. Exemplo de classe: Os seres humanos.
Objeto	É uma instância de uma classe. Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos. Exemplo da classe de objetos da classe humanos: Ana, Pedro.
Atributos	São as características de um objeto. Exemplos: nome, altura, peso, idade, etc. Por sua vez, os atributos possuem valores. Por exemplo, o atributo tipo da seção pode conter o valor azul.
Estado	É o conjunto de valores dos atributos de um determinado objeto.
Métodos	Definem as habilidades dos objetos. Ana é uma instância da classe Pórtico, portanto tem habilidade para adicionar nome do

	caminhar, implementada através do método caminhar (). Em uma classe, um método é apenas uma definição. A ação só ocorre quando o método é invocado através do objeto, no caso Ana. Dentro do programa, a utilização de um método deve afetar apenas um objeto em particular; Todos os seres humanos podem caminhar, mas você quer que apenas Ana caminhe. Normalmente, uma classe possui vários métodos, que no caso da classe seres humanos poderiam ser comer (), dormir() e estudar().
Mensagem	É uma chamada a um objeto para invocar um de seus métodos, ativando um comportamento descrito por sua classe. Também pode ser direcionada diretamente a uma classe (através de uma invocação a um método estático). Em síntese, é a forma que os objetos utilizam para se comunicar.
Abstração	É a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais. Em modelagem orientada a objetos, uma classe é uma abstração de entidades existentes no domínio do sistema de software.
Interface	É um contrato entre a classe e o mundo externo. Quando uma classe implementa uma interface, ela está comprometida a fornecer o comportamento publicado pela interface.

Fonte: Adaptada de Chagas e Oliveira, 2009.

2.6.4.1 Os três fundamentos da Orientação a Objetos

Para se entender propriamente a orientação a objetos, além dos conceitos elementares anteriormente expostos, é importante se atentar para três fundamentos básicos da programação orientada a objetos, que são eles: **encapsulamento**, **herança** e **polimorfismo**.

2.6.4.1.1 Encapsulamento

Encapsulamento pode ser entendido como um agrupamento em um determinado lugar de códigos que estão inseridos em um conjunto dentro do programa e a esse grupo permite-se atribuir um nome. Com isso, surge o conceito de sub-rotina e sempre que se necessitar, é possível requisitar a execução do grupo pelo seu nome atribuído (GOETTEN e WINCK, 2006).

O encapsulamento, afirma Chagas e Oliveira (2009) permite que o programador insira mais segurança, já que o mesmo esconde as propriedades, criando entidades de software estilo a uma caixa preta. A figura 13 mostra essa ideia.

Figura 13 – Representação da caixa preta recebendo mensagens.

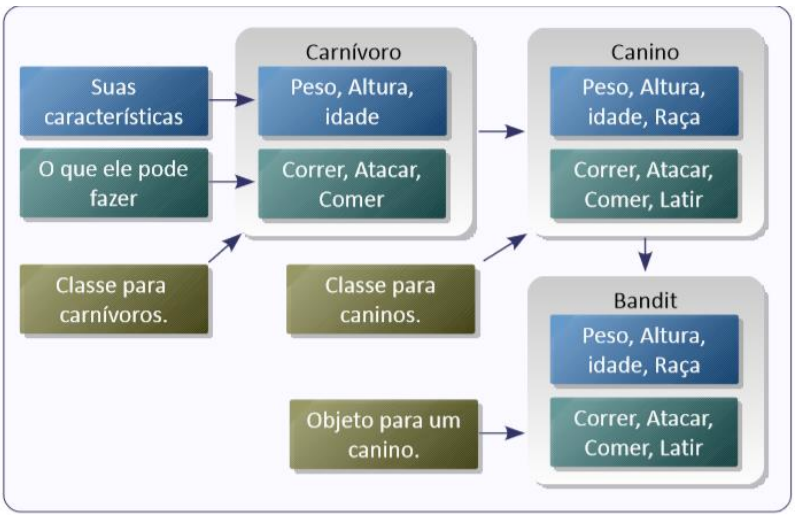
Fonte: Goetten e Winck, 2006.

2.6.4.1.2 Herança

De acordo com Borges (2010), a herança é dita como um mecanismo de orientação a objeto e objetivando a facilidade de reaproveitamento de código,

onde o intuito é que as classes sejam compostas com a formação de uma hierarquia. A figura 14 apresenta um exemplo de como funciona, basicamente, a herança.

Figura 14 – Exemplificação do uso de herança.

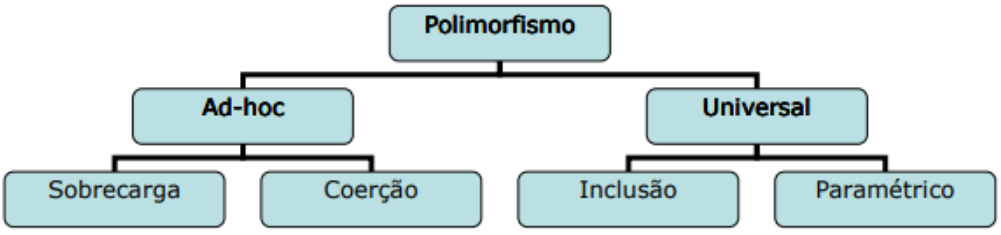


Fonte: Borges, 2010.

2.6.4.1.3 Polimorfismo

O polimorfismo de forma geral significa a capacidade de se apresentar em várias formas. Onde as classes descendentes podem implementar de forma diferente um conjunto de operações comuns definidas na classe base (CHAGAS e OLIVEIRA, 2009). Existem quatro formas de polimorfismo, como observado na figura 15.

Figura 15 – Tipos de polimorfismo.



Fonte: Chagas e Oliveira, 2009.

3 METODOLOGIA

No presente trabalho, realizou-se o desenvolvimento de um software para o dimensionamento de fundação do tipo tubulão com base alargada para utilização em torres de linhas transmissão. Na metodologia deste trabalho será descrito os procedimentos de cálculo utilizados para o dimensionamento e o desenvolvimento do código em Python.

3.1 Caracterização dos materiais empregados

Define-se os parâmetros de materiais empregados:

- Resistência característica do concreto (f_{ck}).
- Resistência característica do aço à tração (f_{yk}).
- Peso específico do concreto (γ_c).

Com base nesses dados pré-determinados, foi possível obter outros parâmetros necessários seguindo equações especificadas pela NBR 6118:2014, como pode-se observar a seguir.

$$f_{cd} = {f_{ck}}/{\gamma_{conc}} \tag{16}$$

$$f_{cfk,inf} = 0{,}21 . f_{ck}^{2/3} \tag{17}$$

$$f_{yd} = {f_{yk}}/{\gamma_s} \tag{18}$$

Onde, γ_{conc} e γ_s possuem valores de 1,4 e 1,15, respetivamente.

3.2 Parâmetros geométricos do tubulão

3.2.1 Determinação do diâmetro da base

Conforme Chaves (2004), o diâmetro da base pode ser determinado por tentativas de maneira a garantir que as tensões horizontais estejam dentro dos

valores admissíveis. O mesmo também pode ser determinado a partir da tensão admissível do solo. Albieiro e Cintra (1996) recomendam para a tensão admissível, para qualquer tipo de solo brasileiro, na prática de projetos a equação (19).

$$\sigma_{adm} = {NSPT}/{5} \quad (19)$$

Com isso, pode-se determinar o diâmetro da base de acordo com a equação (20) proposta por Alonso (2010).

$$D_b = \sqrt{{4P}/{\sigma_{adm}.\pi}} \quad (20)$$

Onde:

P = a força vertical aplicada na fundação;

σ_{adm} = a tensão admissível do solo determinada pela equação 19.

3.2.2 Determinação do diâmetro do fuste

Alonso (2010) relata que o diâmetro do fuste é determinado de maneira análoga ao da base, onde ao invés de se utilizar a tensão admissível do solo, utiliza-se a tensão admissível do concreto. A equação (21) determina a tensão admissível do concreto de acordo com a NBR 6122:1996, no qual os parâmetros γ_c e γ_f possuem valores de 1,6 e 1,4, respectivamente.

$$\sigma_{conc} = {0{,}85.f_{ck}}/{\gamma_c.\gamma_f} \quad (21)$$

Com isso, pode-se determinar o diâmetro do fuste de acordo com a equação (22).

$$D_f = \sqrt{{4P}/{\sigma_{conc}.\pi}} \quad (22)$$

Onde:

P = a força vertical aplicada na fundação;

σ_{conc} = a tensão admissível do concreto determinada pela equação 21.

3.2.3 Determinação da área da base e da área do fuste

O tubulão em estudo possui seção circular em seu fuste e em sua base, desse modo, pode-se determinar a área da base (Ab) e a área do fuste (Af) de acordo com as equações (23) e (24), respectivamente.

$$A_b = {}^{\pi.D_b^2}/_{4} \tag{23}$$

$$A_f = {}^{\pi.D_f^2}/_{4} \tag{24}$$

3.2.4 Determinação da altura do cone

A altura do cone (H) é determinada a partir do diâmetro da base e do fuste e do ângulo de inclinação com a horizontal. Conforme Velloso e Lopes (2010), os alargamentos da base são executados de forma que não necessite de armadura e para isso, adota-se um ângulo de 60º com a horizontal. No qual, essa altura não deve ultrapassar 2 m, de acordo com a NBR 6122:1996. Assim, a equação (25) determina esse parâmetro.

$$H = {}^{(D_b - D_f).tg\ 60^\circ}/_{2} \leq 2\ m \tag{25}$$

3.2.5 Determinação do volume da fundação

De acordo com Alonso (2010) volume da base pode ser calculado como a soma do volume de um cilindro de 20 cm e um “tronco” de cone com altura (H-20). A equação (26) descreve essa afirmação.

$$V_b = 0{,}2.A_b + \left({}^{H - 0{,}2}/_{3}\right)(A_b + A_f + \sqrt{A_b.A_f}) \tag{26}$$

O volume total é determinado pela soma do volume da base com o volume do fuste, conforme a equação (27).

$$V_t = V_b + (\pi.r^2.hf) \tag{27}$$

Onde:

r = raio do fuste;

hf = altura do fuste.

3.2.6 Determinação da área de aço mínima em um tubulão

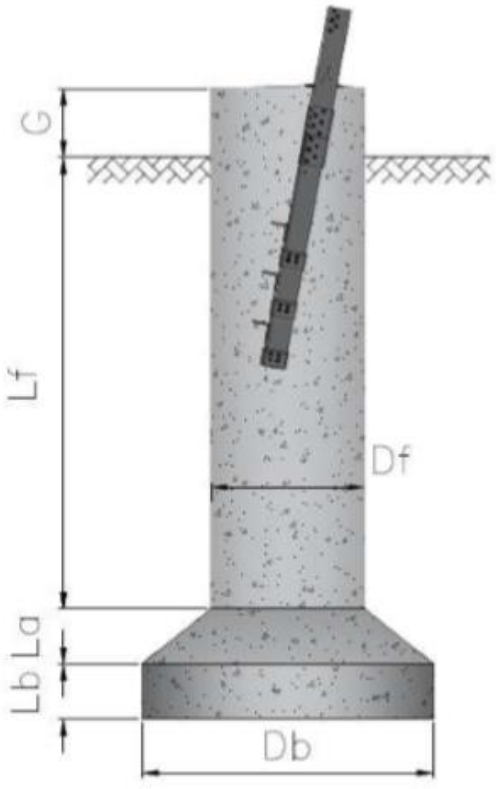
De acordo com a NBR 6122 a taxa de aço nos tubulões não deve ser inferior a 0,5% da seção necessária, com isso temos a equação (28).

$$A_s = 0{,}005 . {}^{(\pi . D_f^2)}/_{4} \quad (28)$$

3.3 Procedimento de cálculo geotécnico e estrutural de fundações do tipo tubulão

A figura 16 mostra um esquemático do tubulão com base alargada e suas respectivas dimensões.

Figura 16 – Esquemático do tubulão com base alargada.



Fonte: Amaral, 2015.

3.3.1 Verificação à compressão

A tensão atuante de cálculo deve ser menor ou igual a tensão admissível do solo, conforme as equações a seguir citadas no item 3.5.1 deste trabalho.

$$q_s = \frac{P + G_{tub} - A_l f_{adm}}{Ab} \leq q_{adm}$$

Onde:

$$q_{adm} = \frac{1}{3}(cN_C S_C + 0{,}5\gamma B N_\gamma S_\gamma + qN_q S_q)$$

3.3.2 Verificação ao arrancamento

Para análise ao arrancamento utilizou-se o método do cone invertido descrito no item 3.5.2, no qual tem-se a equação (29):

$$G_{tub} + P_s \geq Fa.FS \qquad (29)$$

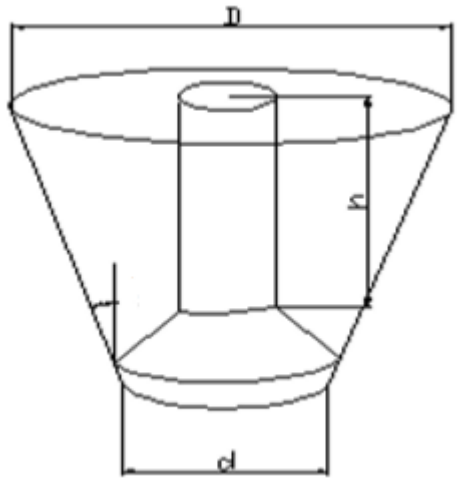
Onde:

Fa = força de arrancamento;

FS = 1,10.

Para determinação do volume de terra segue-se a equação (30) e (31) a seguir, observando os parâmetros na figura 17.

Figura 17 – Parâmetros para o método do cone.



Fonte: Quental, 2008.

$$D = d + 2.h.tg \quad (30)$$

$$Vt = \left(\pi.{}^{h}/_{12}\right).\left(D^2 + Dd + d^2\right) - Vf \quad (31)$$

Onde:

Vt = volume de terra;

Vf = volume da fundação (equivalente ao volume de concreto);

D = diâmetro maior do tronco de cone;

d = diâmetro da base da fundação do tronco de cone (equivalente ao Db);

α = ângulo de inclinação (20º para areias e 30º para argilas);

h = altura do fuste da fundação.

3.3.3 Cálculo estrutural da fundação

A armadura irá resistir os esforços de flexo-compressão e flexo-tração devido aos esforços verticais e horizontais aplicados na fundação, onde essa armadura é distribuída de forma simétrica ao longo do fuste. Para determinação da hipótese que exige maior taxa de armadura (ω), utilizou-se o ábaco 6.2. de Pfeil (1989), que se encontra no Anexo A. Os coeficientes utilizados como entrada são o ν e μ, a determinação dos mesmos dependem das equações (32) e (33).

$$\nu = {}^{N_d}/_{(f_c.D_f)} \quad (32)$$

$$\mu = \nu.\left({}^{e}/_{D_f}\right) \quad (33)$$

Onde *e* representa a excentricidade sendo calculada pela equação (34).

$$e = {}^{M_d}/_{N_d} \quad (34)$$

Temos que M_d é o momento de cálculo e é definido pela equação (35).

$$M_d = H\left(G + 0{,}545\sqrt{{}^{H}/_{\left[\gamma' + D_f.\tan^2\left(45 + \frac{\varphi}{2}\right)\right]}}\right) \quad (35)$$

Onde:

H = força horizontal aplicada na fundação;

G = afloramento do tubulão;

γ' = peso específico efetivo do solo;

φ = ângulo de atrito do solo.

3.3.3.1 Determinação da área de aço da armadura longitudinal do tubulão

A área de aço é determinada pela NBR 6118:2014 pela equação (36) apresentada a seguir, onde a mesma depende do ρ que é determinado seguindo a equação (37).

$$A_s = \rho \left(\pi . D_f^2 / 4 \right) \quad (36)$$

$$\rho = \omega \left(f_c / f_s \right) \quad (37)$$

Onde ω é a taxa de aço determinada pelo ábaco 6.2 de Pfeil (1989).

A partir da área de aço determina-se o diâmetro da barra longitudinal a ser utilizada seguindo procedimento abaixo.

1. Calcula-se a área de aço total para o tubulão;
2. Determina a área de aço unitária pela razão entre a área de aço total e número de barras longitudinais, onde o número de barras é pré-determinado pelo arranjo escolhido, no qual neste trabalho o arranjo foi de 18 barras seguindo o ábaco 6.2 de Pfeil (1989);
3. Com a área unitária é determinado o diâmetro da barra a partir tabela de área aço (Anexo B).

3.3.3.2 Determinação da armadura transversal do tubulão

O diâmetro da armadura transversal é definido pela NBR 6118:2014 no item 18.4.3., onde o mesmo prescreve o seguinte para o diâmetro:

$$\phi_w \geq \begin{cases} 5\ mm \\ \phi_l / 4 \end{cases} \quad (38)$$

3.3.3.2.1 Determinação do espaçamento da armadura transversal

O espaçamento da armadura transversal é definido pela NBR 6118:2014 no item 18.4.3., visto que no presente trabalho utilizou-se CA 25 para a armadura transversal, tem-se que:

$$s_w \leq \begin{cases} 200\ mm \\ D_f \\ 24*CA\ 25 \end{cases} \quad (39)$$

Com o diâmetro da armadura transversal e seu espaçamento, torna-se possível determinar a área de aço da mesma, na qual é baseada no procedimento descrito a seguir:

1. Calcula-se a quantidade de armadura transversal que será disposta ao longo do fuste pela razão entre a altura do fuste e o espaçamento transversal;
2. Determina-se a área de uma armadura unitária com o seu diâmetro encontrado pelo cálculo da área de um círculo $\left(\pi.\phi_w^2 / 4\right)$;
3. Com o número de armaduras transversais pode-se obter a área de aço transversal total pelo produto entre o número de armadura e a área da armadura unitária.

3.3.4 Verificação a compressão diagonal e da armadura transversal

As verificações da seção transversal do fuste são feitas utilizando-se o modelo I de cálculo, prevista pela NBR-6118:2014, no item 17.4.2.2.. O modelo I admite diagonais de compressão inclinadas de θ = 45° em relação ao eixo longitudinal do elemento estrutural. Para que haja garantia de segurança da estrutura mediante os esforços transversais, deve-se respeitar, simultaneamente, as duas seguintes condições:

$$V_{sd} \leq V_{rd} \quad \textbf{(condição 1)} \tag{40}$$

$$V_{rd} = 0{,}27.\alpha_{u2}.f_{cd}.b_w.d \tag{41}$$

$$\alpha_{u2} = 1 - {}^{f_{ck}}\!/_{250} \tag{42}$$

$$V_{sd} \leq V_c + V_{sw} \quad \textbf{(condição 2)} \tag{43}$$

$$V_{sw} = \left({}^{A_{sw}}\!/_{s}\right).0{,}9.d.f_{yd} \tag{44}$$

$$V_c = \left({}^{f_{ctk,inf}}\!/_{\gamma_{conc}}\right).b_w.d \tag{45}$$

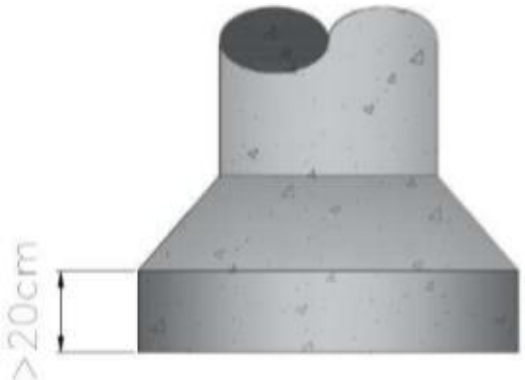
3.3.5 Base alargada do tubulão

A base do tubulão será determinada bem como um bloco de concreto simples, sem armação. Visto que a transição do fuste para a base é feita seguindo a superfície tronco-cônica do alargamento, ou seja:

$$D_b + D_f \leq {}^{\tan(60°)}\!/_{2} \tag{46}$$

Vale salientar que a altura do rodapé da base tem no mínimo 20 cm, conforme a NBR 6122:1996, onde foi o valor adotado para os cálculos desenvolvidos neste trabalho. A figura 18 mostra a representação da base alargada.

Figura 18 – Base alargada do tubulão.



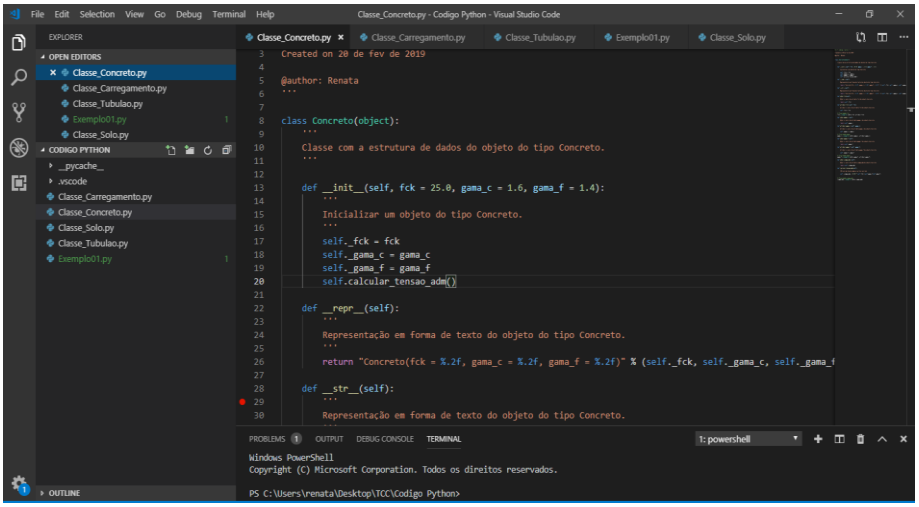
Fonte: Campos, 2015.

3.4 Visual studio code

A princípio determinou-se o programa no qual fosse possível utilizar a linguagem Python e desenvolvê-la com facilidade. O programa utilizado foi Visual Studio Code (vide figura 19), em que o mesmo consiste, basicamente, em um editor de código-fonte. A partir dele, torna-se possível a elaboração de projetos, como a criação de programas, geração de bancos de dados e isso, em diversas linguagens de programação, dentre elas, o Python.

Para elaboração de todo o programa fez-se uso de diversas bibliotecas, para representação gráfica e solução de problemas matemáticos, onde destacam-se a math, numpy e a plotly.

Figura 19: Interface do programa Visual Studio Code.



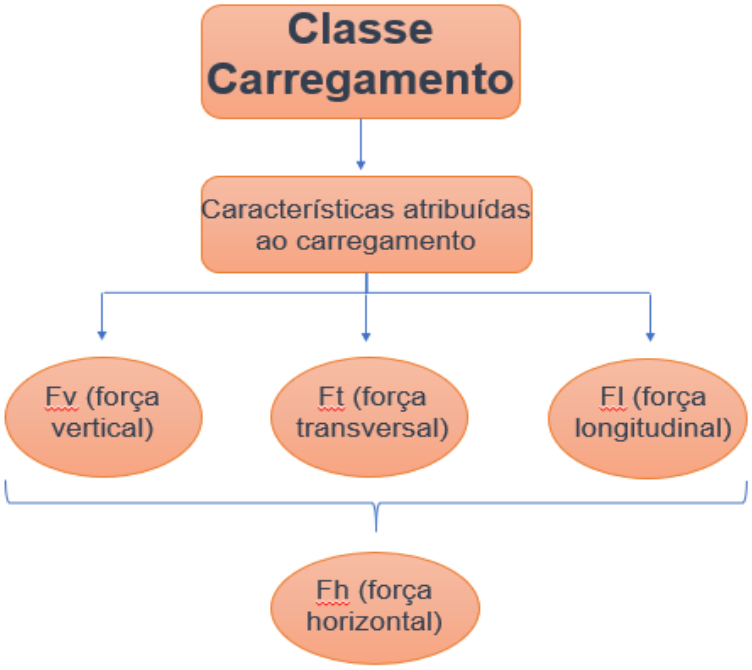
Fonte: Autoria própria.

3.5 Determinação das classes

Com o procedimento de cálculo e o programa a ser utilizado definido, prosseguiu-se com a determinação das classes que são estruturas básicas representando o tipo do objeto e contendo suas características e ações, como mencionado no item 3.6 deste trabalho. Com as características e efeitos determinados, torna-se possível dar início ao programa e realizar uma representação gráfica do mesmo.

Dessa forma, desenvolveu-se as seguintes classes: carregamento, concreto, solo e tubulão. Na classe de carregamento está inserido os esforços que a torre, provavelmente, será submetida, que são esforços verticais, transversais, longitudinal e horizontal. Onde, o valor do esforço vertical, transversal e longitudinal são valores atribuídos, já o esforço horizontal é o vetor resultante da força longitudinal e transversal. A figura 20 e 21 apresenta o diagrama da classe carregamento e a interface da classe de carregamento, respectivamente, e seu código-fonte está inserido do Anexo C.

Figura 20 – Diagrama da classe carregamento.



Fonte: Autoria própria.

Figura 21 – Interface da classe de carregamento.

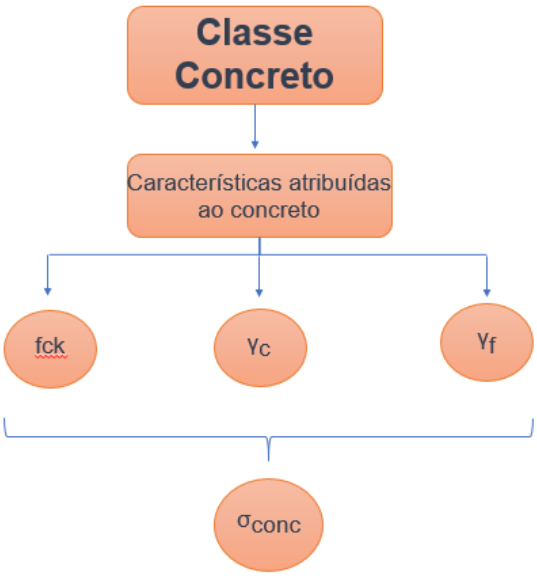
Classe_Concreto.py | apagar.txt | Classe_Carregamento.py | Classe_Tubulao.py | settings.json | drawTubular.py

```
# -*- coding: latin1 -*-
'''
Created on 28 de fev de 2019

@author: Renata
'''

class Carregamento (object):

    '''
    Classe com a estrutura de dados do objeto do tipo Carregamento.
    '''

    def __init__(self, fv= 0.0, ft= 0.0, fl= 0.0):
        '''
        Inicializar um objeto do tipo Carregamento.
        '''
        self._fv = fv
        self._ft = ft
        self._fl = fl
        self.calcular_fh()

    def __repr__(self):
        '''
        Representação em forma de texto do objeto do tipo Carregamento.
        '''
        return "Carregamento(fv = %.2f, ft = %.2f, fl = %.2f, fh = %.2f)" % (self.fv,self.ft,self.fl,self.fh)

```

Fonte: Autoria própria.

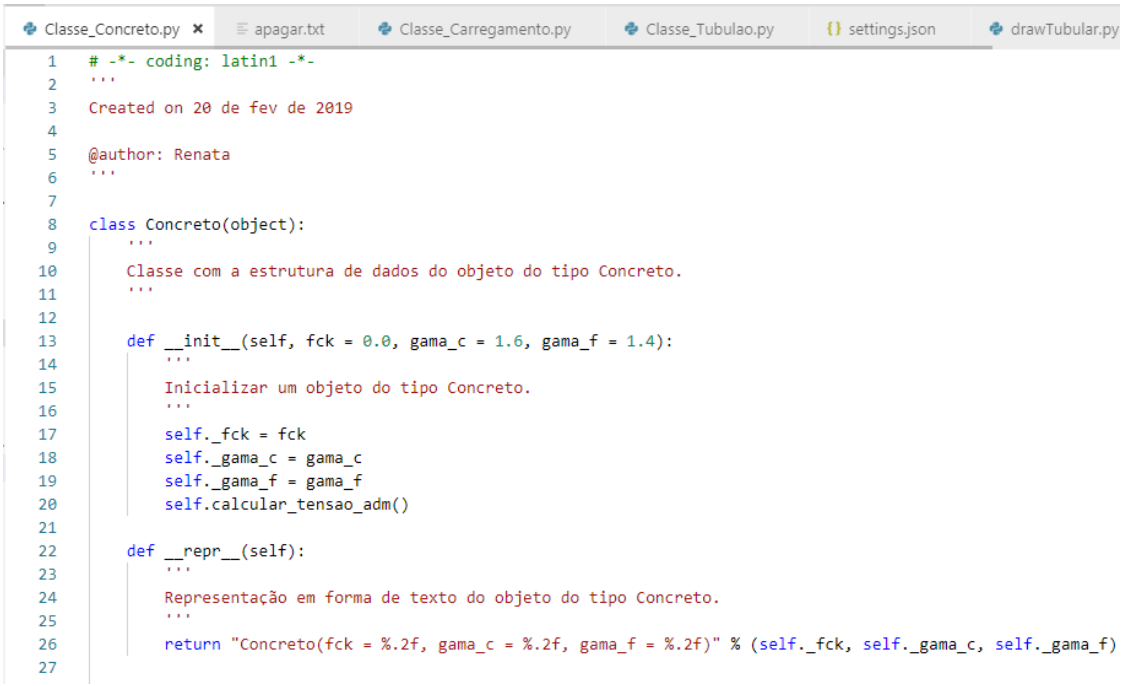
Na classe concreto atribuiu-se os parâmetros de f_{ck}, γ_c e γ_f, para em seguida determinar a tensão admissível do concreto e com base nos resultados iniciar a determinação dos parâmetros geométricos do tubulão. A figura 22 e 23 apresenta o diagrama da classe concreto e interface da classe concreto, respectivamente, e o seu código-fonte está inserido no Anexo D.

Figura 22: Diagrama da classe Concreto.



Fonte: Autoria própria.

Figura 23 – Interface da classe concreto.



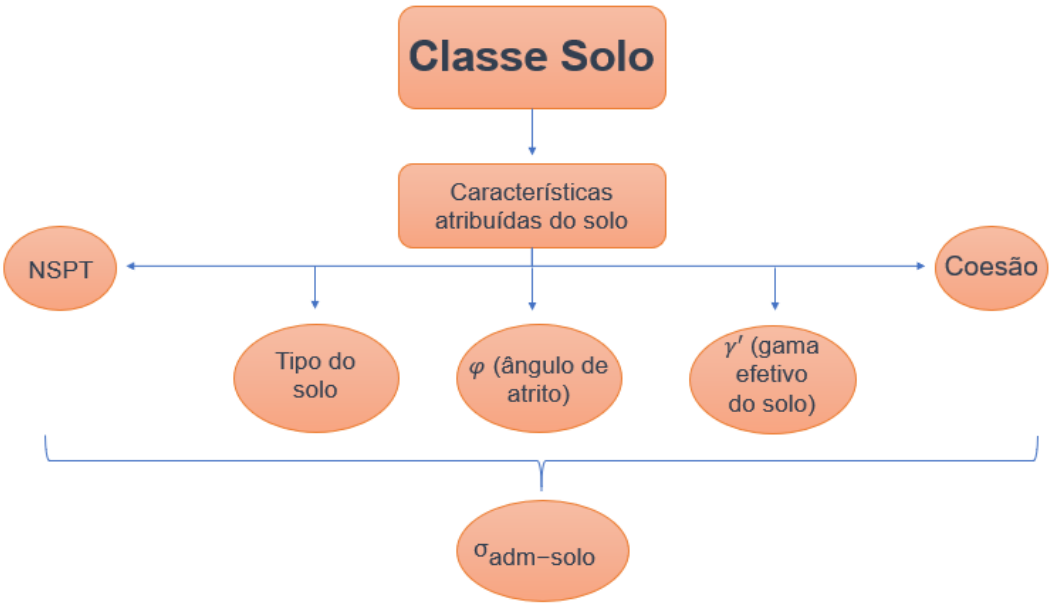

```python
# -*- coding: latin1 -*-
'''
Created on 20 de fev de 2019

@author: Renata
'''

class Concreto(object):
    '''
    Classe com a estrutura de dados do objeto do tipo Concreto.
    '''

    def __init__(self, fck = 0.0, gama_c = 1.6, gama_f = 1.4):
        '''
        Inicializar um objeto do tipo Concreto.
        '''
        self._fck = fck
        self._gama_c = gama_c
        self._gama_f = gama_f
        self.calcular_tensao_adm()

    def __repr__(self):
        '''
        Representação em forma de texto do objeto do tipo Concreto.
        '''
        return "Concreto(fck = %.2f, gama_c = %.2f, gama_f = %.2f)" % (self._fck, self._gama_c, self._gama_f)

```

Fonte: Autoria própria.

Na classe solo atribuiu-se o valor do NSPT, para em seguida determinar a tensão admissível do solo e com base nos resultados determinar parâmetros geométricos do tubulão. Também se inseriu outros parâmetros de entrada que são característicos ao solo, como: a coesão, o ângulo de atrito representado por *phi*, o peso específico do solo representado por *gama_solo* e o tipo de solo onde será implantada a fundação, no qual está representado pela variável *t.* A figura 24 e 25 apresentam o diagrama e a interface da classe solo, respectivamente, e o seu código-fonte está inserido no Anexo E.

Na classe tubulão importou-se as classes anteriores para obter os valores calculados dentro das mesmas e definir todos os parâmetros geométricos, geotécnicos e estruturais citados nos itens 4.2 e 4.3, bem como suas verificações que validam a geometria calculada. A figura 26 e 27 apresenta o diagrama e a interface da classe tubulão, respectivamente, e seu código-fonte está inserido no Anexo F.

Figura 24 – Diagrama da classe solo.



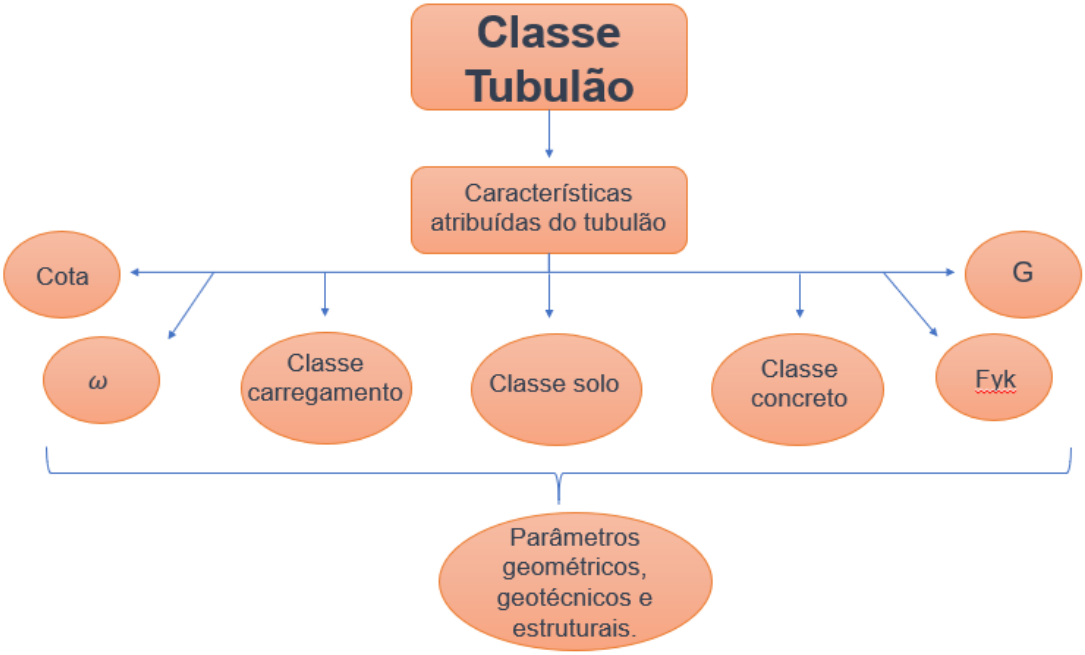
Fonte: Autoria própria.

Figura 25 - Interface da classe solo.

```
# -*- coding: latin1 -*-
'''
Created on 24 de fev de 2019

@author: Renata
'''
import math
import numpy

#from Classe_Tubulao import Tubulao

class Solo (object):

    '''
    Classe com a estrutura de dados do objeto do tipo Solo.
    '''

    def __init__(self, nspt= 0.0,c= 0.0,phi= 0.0, gama_solo = 0.0, sy= 0.60, t = "areia"):
        '''
        Inicializar um objeto do tipo Solo.
        '''
        self._nspt = nspt
        self._c = c
        self._phi = phi
        self._gama_solo = gama_solo
        self._sy = sy
        self.calcular_tensao_adm()
        self._tipo = t
        if t == "areia":
```

Fonte: Autoria própria.

Figura 26 – Diagrama da classe tubulão.



Fonte: Autoria própria.

Figura 27 – Interface da classe tubulão.

Classe_Carregamento.py | Classe_Tubulao.py | settings.json | drawTubular.py | Exemplo01.py | Exemplo02.py | Classe_S

```
1  # -*- coding: utf-8
2  '''
3  Created on 28 de fev de 2019
4
5  @author: Renata
6  '''
7  import math
8  import numpy
9
10 from Classe_Concreto import Concreto
11 from Classe_Solo import Solo
12 from Classe_Carregamento import Carregamento
13
14 class Tubulao(object):
15
16     '''
17     Classe com a estrutura de dados do objeto do tipo Tubulão.
18     '''
19
20     def __init__(self, cota= 0.0, g = 0.0, w = 0.0, fyk = 500.0, carregamento = None, concreto = None, solo = None):
21         '''
22         Inicializar um objeto do tipo Tubulão.
23         '''
24         self._cota = cota
25         self._g = g
26         self._w = w
27         self.fyk = fyk
28         self._carregamento = carregamento
29         self._concreto = concreto
30         self._solo = solo
```

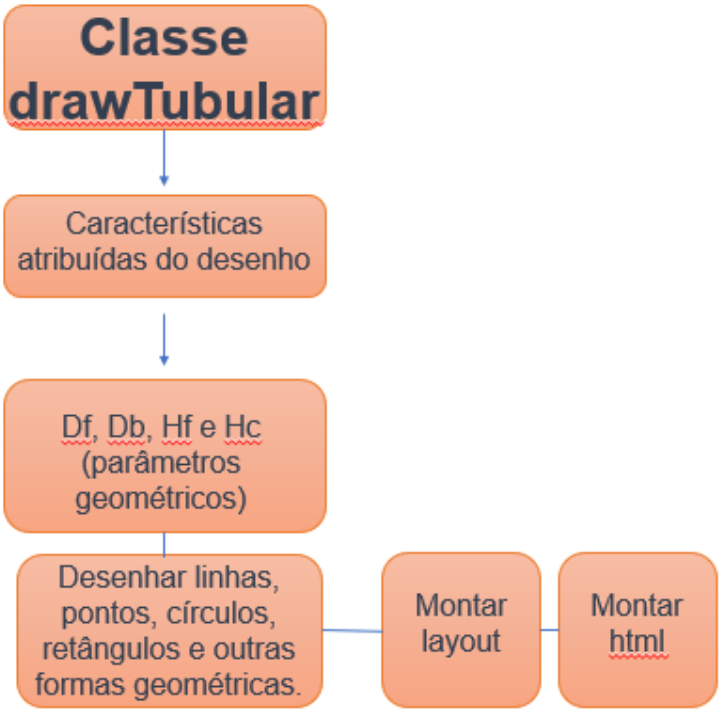
Fonte: Autoria própria.

Com todos os parâmetros de entrada e de cálculo determinados, seguiu-se para execução do código para o desenho do tubulão com suas respectivas

dimensões. Para a implementação do desenho em 2D utilizou-se a biblioteca *plotly*, que pode gerar vários tipos de gráficos interativos, dentre outros. Também se utilizou a biblioteca *dash* para a representação visual em 2D aliada a *plotly*.

Na sequência, criou-se a classe drawTubular para o desenvolvimento do código referente ao desenho. Na figura 28 e 29 pode ser observado o diagrama e a interface da classe drawTubular, respectivamente, seu código-fonte está inserido no Anexo G.

Figura 28 – Interface da classe tubulão.



Fonte: Autoria própria.

Figura 29 – Interface da classe drawTubular.

Classe_Carregamento.py | Classe_Tubulao.py | settings.json | drawTubular.py

```python
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objs as go

import numpy as np

class tubular():
    #Todas as unidades estão dadas em cm
    def __init__(self, alturaCone, alturaTotal, dF, dB):
        self.alturaCone = alturaCone
        self.alturaTotal = alturaTotal
        self.dF = dF#Diâmetro do fuste
        self.dB = dB#Diâmetro da base
        self.lb = 20#largura do rodapé
        self.la = alturaCone-self.lb
        self.lf = alturaTotal - alturaCone
        self.shapes = []#Os desenhos que foram feitos
        self.app = dash.Dash()
        self.xRange = [0,self.alturaTotal*1.1+1.6*self.dB]
        self.yRange = (-1.6*self.dB,self.alturaTotal*1.3)
        self.figureSizeX = self.xRange[0]+self.xRange[1]
        self.figureSizeY = np.abs(self.yRange[0])+self.yRange[1]

    def desenharTubular(self,pos,**kwargs):···
```

Fonte: Autoria própria.

3.6 Desenvolvimento do desenho em 2D

Para o desenvolvimento do desenho, fez uso das bibliotecas dash, plotly e numpy. A biblioteca dash é utilizada neste trabalho para obter um resultado mais dinâmico e em tempo real. Porém, sua função propriamente dita, seria unir a biblioteca dash com a plotly e realizar a geração de uma guia, ou seja, a interface gráfica do programa que não foi objetivada neste projeto. A biblioteca ploty foi para implementação de todo o desenho em 2D e seus parâmetros necessários, já a numpy se encontra em todos as classes já que a mesma tem função permite trabalhar com diversas funções e operações avançadas.

Para o desenvolvimento geométrico do programa foram utilizadas algumas formas como: shape, line, circle e rectangle. Os parâmetros de posicionamento foram ajustados manualmente, visando a localização mais organizada possível.

Inseriu-se todos os componentes referentes a geometria do tubulão para a representação no desenho, são eles: Df, Db, Hc, Cota e a representação da

armadura do fuste. Com isso, desenvolveu-se o código para desenhar os seguintes itens:

- Desenhar armadura;
- Desenhar círculos;
- Desenhar linhas;
- Desenhar pontos;
- Desenhar furos;

Logo após, determinou-se a cor na qual o desenho iria ser gerado e inserido os textos necessários na aplicação do desenho. Vale salientar, que para cada desenho solicitado, foi aplicado pontos de início e fim para o posicionamento da figura. Para a realização do desenho da seção transversal do fuste, foi utilizado markers que é um estilo de representação em pontos.

Por fim, foi feito a montagem do layout e do arquivo em html, onde será gerado o desenho em 2D. O código-fonte referente ao desenho encontra-se no Anexo E.

4 RESULTADOS E DISCUSSÕES

Nesta seção será validado as resultas do programa desenvolvido para aplicação em fundações do tipo tubulão. Para avaliar a aplicabilidade desta ferramenta, foram desenvolvidos 2 exemplos.

4.1 Testes e aplicações

4.1.1 Exemplo 1

Para o exemplo 1, temos os parâmetros de entrada dispostos na tabela 2 e suas respectivas unidades de entrada no programa. Pode-se observar também a seguir, para o exemplo 1, o script produzido para execução do problema.

Tabela 2 – Parâmetros de entrada para o exemplo 1.

Cota (assentamento do tubulão – cm)	600
Fck (resistência característica do concreto – MPa)	14
NSPT (número de golpes médio)	15
C (coesão – kN/m³)	0
φ (ângulo de atrito do solo)	20º
γ' (gama efetivo do solo – kN/m³)	17
Tipo do solo (t)	areia
Fv (força vertical – kN)	400
Ft (força transversal – kN)	300
Fl (força longitudinal – kN)	200

Fonte: Autoria própria.

SCRIPT – EXEMPLO 1

```
# -*- coding: utf-8


from Classe_Tubulao import Tubulao
from Classe_Concreto import Concreto
from Classe_Solo import Solo
from Classe_Carregamento import Carregamento
#desenho
from drawTubular import tubular

fundacao01 = Tubulao(cota = 600,g = 0)

fundacao01.concreto = Concreto(fck = 14.0)
fundacao01.solo = Solo(nspt=15,c=0,phi=20,gama_solo=17,t="areia")
fundacao01.carregamento = Carregamento(fv=400,ft=300,fl=200)

#fundacao01.processar()

print("O valor do Df = %.2f" % fundacao01.df)
print("O valor do Db = %.2f" % fundacao01.db)
print("O valor do Hcone = %.2f" % fundacao01.hc)
print("O valor do Hfuste = %.2f" %fundacao01.hf)
print("O valor da Abase = %.2f" %fundacao01.ab)
print("O valor da Afuste = %.2f" %fundacao01.af)
```

```
print("O valor do Vb = %.2f" %fundacao01.vb)
print("O valor do Vf = %.2f" %fundacao01.vf)
print("O valor do Vt = %.2f" %fundacao01.vt)
print("O valor da As = %.2f" %fundacao01.asm)
print("O valor do Md = %.2f" %fundacao01.md)
print("O valor do e = %.2f" %fundacao01.e)
print("O valor do v = %.2f"%fundacao01.v)
print("O valor do m = %.2f"%fundacao01.m)
print("O valor da Ast = %f"%fundacao01.ast)
print("Arranjo de barras: 18 d %.2f"%fundacao01.diametro_barra)
fundacao01.verificacao_comp_diag()
fundacao01.verificacao_compressao()
fundacao01.verificacao_arrancamento()
fundacao01.verificacao_armad_trans()
print ("Diametro da armadura transversal :
%.f"%fundacao01.armadura_transv())
print ("Espaçamento da armadura transversal :
%.f"%fundacao01.espacamento_w())


t = tubular(fundacao01.hc,fundacao01.hc+fundacao01.hf,fundacao01.df,
fundacao01.db)
t.desenhar()
```

Com os parâmetros definidos e inseridos no programa, foi possível executá-lo e obter os resultados, que estão descritos na tabela 3. A figura 30 apresenta o interpretador com os resultados apresentados na tabela 3.

Tabela 3 – Parâmetros geométricos calculados pelo programa – exemplo 1.

Df (cm)	70
Db (cm)	131,55
Hc (cm)	53,30
Hf (cm)	546,70
Ab (cm²)	13591,57
Af (cm²)	3848,45
Vt (cm³)	2380538,54
Asm (cm²)	19,24
Md (kN.m)	295,19

e (m)	0,58
ν	0,02
μ	0,02

Fonte: Autoria própria.

A partir do resultado dos parâmetros ν e μ, obtidos pelo programa e com base no procedimento descrito no item 4.3.3.1 deste trabalho, obteve-se um valor de ω = 0,1 no ábaco 6.2. Pfeil (1989), com isso os valores calculados pelo programa estão dispostos na tabela 4.

Tabela 4 – Parâmetros de cálculo estrutural armadura longitudinal – exemplo 1.

Ast (cm²)	19,24
Arranjo das barras	18 ϕ 12,5 mm

Fonte: Autoria própria.

Foi obtido também o diâmetro da armadura longitudinal e seu espaçamento seguindo o procedimento do item 4.3.3.2 contido neste trabalho, em que seus valores podem ser observados na tabela 5.

Tabela 5 – Parâmetros de cálculo estrutural armadura transversal – exemplo 1.

ϕ_w	5 mm
s_w	20 cm

Fonte: Autoria própria.

As verificações definidas no item 4.3 deste trabalho foram executadas de maneira satisfatória e obteve-se os resultados dispostos na tabela 6.

Tabela 6 – Resultados gerados pelo programa para as verificações – exemplo 1.

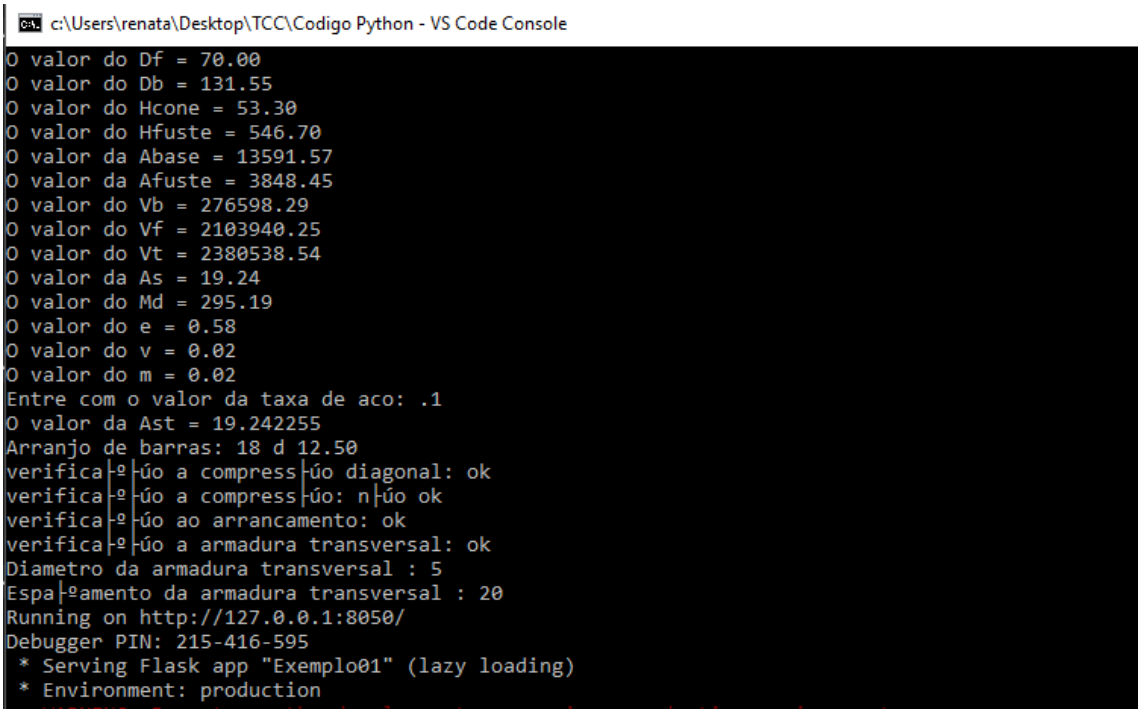
Verificação a compressão	Ok
Verificação ao arrancamento	Ok
Verificação a compressão diagonal	Ok
Verificação da armadura transversal	Ok

Fonte: Autoria própria.

Pode-se observar a partir da figura 30 que o programa calculou satisfatoriamente os resultados dos parâmetros geométricos para o dimensionamento ao tubulão. Bem como, para determinação dos cálculos geotécnicos e estruturais do mesmo.

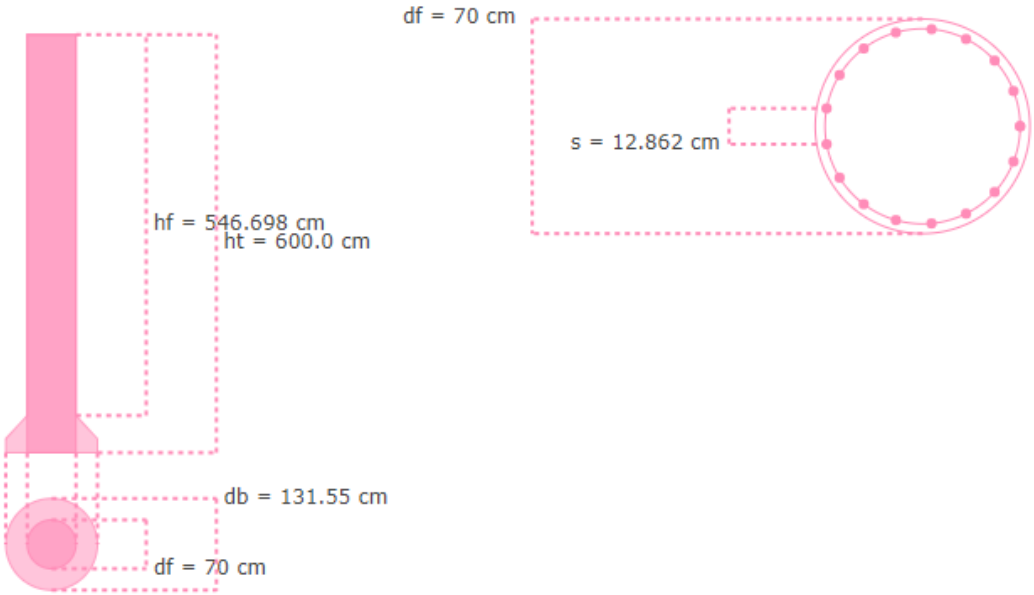
Com todos os dados estabelecidos, torna-se possível a execução do desenho em 2D produzido pelo programa e facilitando o entendimento e análise dos parâmetros de cálculo. A figura 30 mostra no final dos resultados um **running on**, no qual é disponibilizado um link que possibilita gerar o desenho em dash de maneira dinâmica e online. A figura 31 apresenta o desenho em 2D produzido pelo programa.

Figura 30 – Resultados gerados com a execução do programa para o exemplo 1.



Fonte: Autoria própria.

Figura 31: Resultado do desenho em 2D gerado pelo programa para o exemplo 1.



Fonte: Autoria própria.

4.1.2 Exemplo 2

Para o exemplo 2, de maneira análoga ao exemplo 1, temos os parâmetros de entrada dispostos na tabela 7 e suas respectivas unidades de entrada no programa. Observa-se também o script gerado para a execução do programa.

SCRIPT – EXEMPLO 2

```
# -*- coding: utf-8

from Classe_Tubulao import Tubulao
from Classe_Concreto import Concreto
from Classe_Solo import Solo
from Classe_Carregamento import Carregamento
#desenho
from drawTubular import tubular
fundacao01 = Tubulao(cota = 800, g = 0)
fundacao01.concreto = Concreto(fck = 20.0)
fundacao01.solo = Solo(nspt=20,c=30,phi=0,gama_solo=19,t="argila")
fundacao01.carregamento = Carregamento(fv=1000,ft=900,fl=800)

#fundacao01.processar()

print("O valor do Df = %.2f" % fundacao01.df)
print("O valor do Db = %.2f" % fundacao01.db)
print("O valor do Hcone = %.2f" % fundacao01.hc)
print("O valor do Hfuste = %.2f" %fundacao01.hf)
print("O valor da Abase = %.2f" %fundacao01.ab)
print("O valor da Afuste = %.2f" %fundacao01.af)
print("O valor do Vb = %.2f" %fundacao01.vb)
print("O valor do Vf = %.2f" %fundacao01.vf)
print("O valor do Vt = %.2f" %fundacao01.vt)
print("O valor da As = %.2f" %fundacao01.asm)
print("O valor do Md = %.2f" %fundacao01.md)
print("O valor do e = %.2f" %fundacao01.e)
print("O valor do v = %.2f"%fundacao01.v)
print("O valor do m = %.2f"%fundacao01.m)
print("O valor da Ast = %f"%fundacao01.ast)
print("Arranjo de barras: 18 d %.2f"%fundacao01.diametro_barra)
fundacao01.verificacao_comp_diag()
fundacao01.verificacao_compressao()
fundacao01.verificacao_arrancamento()
fundacao01.verificacao_armad_trans()
print("Diametro da armadura transversal :
%.f"%fundacao01.armadura_transv())
```

```
print("Espaçamento da armadura transversal :
%.f"%fundacao01.espacamento_w())
print(fundacao01.tensao_base)
print(fundacao01.tensao_ult)

t = tubular(fundacao01.hc,fundacao01.hc+fundacao01.hf,fundacao01.df,
fundacao01.db)
t.desenhar()
```

Tabela 7 – Parâmetros de entrada para o exemplo 2.

Cota (assentamento do tubulão – cm)	800
Fck (resistência característica do concreto – MPa)	20
NSPT (número de golpes médio)	20
C (coesão – kN/m³)	30
φ (ângulo de atrito do solo)	0º
γ' (gama efetivo do solo – kN/m³)	18
Tipo do solo (t)	argila
Fv (força vertical – kN)	1000
Ft (força transversal – kN)	900
Fl (força longitudinal – kN)	800

Fonte: Autoria própria.

Os valores dispostos na tabela 7 representam os parâmetros de entrada no programa, permitindo assim, sua execução. Os resultados obtidos para o exemplo 2, estão apresentados na tabela 8. A figura 32 apresenta o interpretador com os resultados expostos na tabela 6.

Tabela 8 – Parâmetros geométricos calculados pelo programa para o exemplo 2.

Df (cm)	70
Db (cm)	180,13
Hc (cm)	95,37
Hf (cm)	704,63

Ab (cm²)	25484,2
Af (cm²)	3848,45
Vt (cm³)	990888,81
Asm (cm²)	19,24
Md (kN.m)	2413,95
e (m)	1,43
ν	0,03
μ	0,05

Fonte: Autoria própria.

Analogamente, com o ν e μ calculado, e seguindo o procedimento do item 4.3.3.1 deste trabalho, obteve-se um valor de ω = 0,2 no ábaco 6.2. Pfeil (1989), com isso os valores obtidos pelo programa estão dispostos na tabela 9.

Tabela 9 – Parâmetros de cálculo estrutural para exemplo 2.

Ast (cm²)	30,78
Arranjo das barras	18 ϕ 16 mm

Fonte: Autoria própria.

Obteve-se o diâmetro da armadura longitudinal e seu espaçamento seguindo o procedimento do item 4.3.3.2 contido neste trabalho, em que seus valores podem ser observados na tabela 10.

Tabela 10 – Parâmetros de cálculo estrutural armadura transversal – exemplo 2.

ϕ_w	5 mm
s_w	20 cm

Fonte: Autoria própria.

As verificações descritas no item 4.3 deste trabalho foram realizadas de maneira satisfatória pelo programa, como pode-se observar na tabela 11 e na figura 32.

Tabela 11 – Resultados gerados pelo programa para as verificações – exemplo 2.

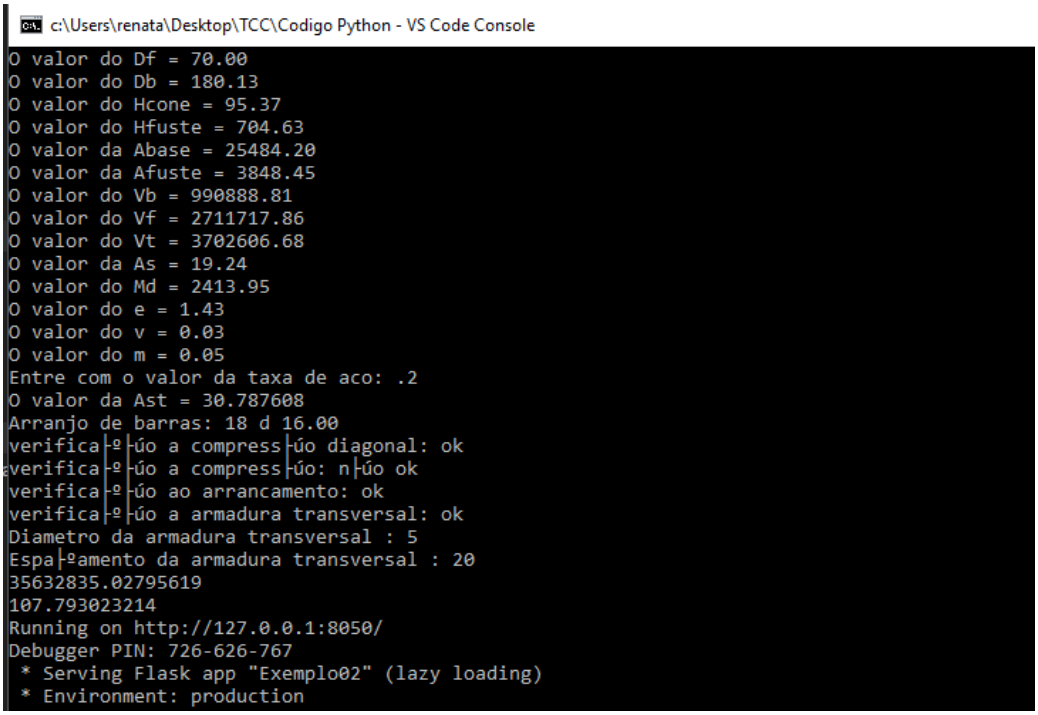
Verificação a compressão	Ok
Verificação ao arrancamento	Ok
Verificação a compressão diagonal	Ok
Verificação da armadura transversal	Ok

Fonte: Autoria própria.

Com a execução do programa foi constatado resultados satisfatórios para os dados geométricos e estruturais direcionados ao dimensionamento do tubulão. A figura 32 mostra os resultados gerados pelo programa.

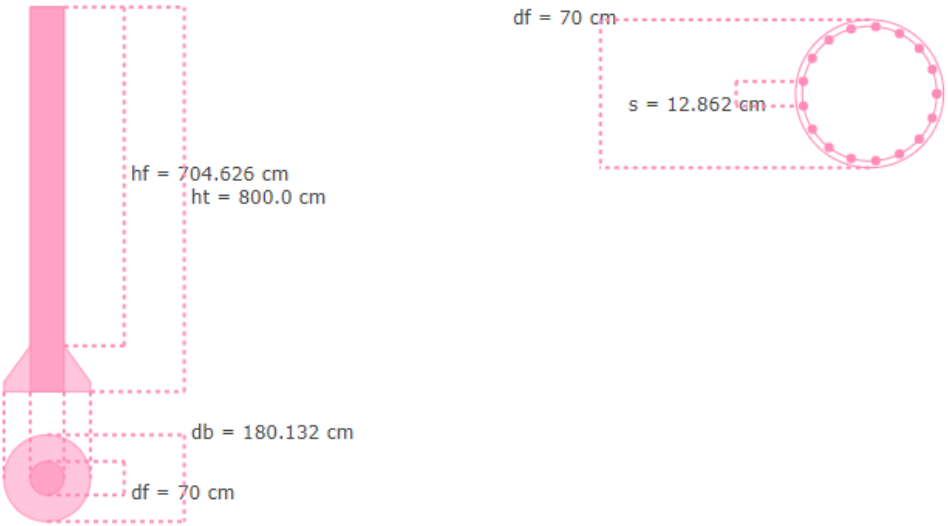
De maneira similar, ao procedimento do exemplo 1, a execução do desenho em 2D produzido pelo programa e facilitando o entendimento e análise dos parâmetros de cálculo. A figura 32 mostra no final dos resultados um “running on”, no qual é disponibilizado um link que possibilita gerar o desenho em dash de maneira dinâmica e online. A figura 33 apresenta o desenho em 2D produzido pelo programa.

Figura 32 – Resultados gerados com a execução do programa para o exemplo 2.



Fonte: Autoria própria.

Figura 33: Resultado do desenho em 2D gerado pelo programa para o exemplo 2.



Fonte: Autoria própria.

5 CONSIDERAÇÕES FINAIS

De acordo com as testes e aplicações feitas, foi possível obter informações importantes através dos resultados encontrados. Os resultados apresentados e as discussões pertinentes conduziram às seguintes conclusões:

- Foi alcançado de maneira satisfatória a realização do software proposto no presente trabalho, bem como, sua perfeita execução;
- No decorrer do trabalho foi possível constatar, a partir de dados bibliográficos, que as principais fundações utilizadas em torres de linhas de transmissão são as sapatas e tubulões devido a sua garantia quanto a eficiência e viabilidade de execução, no qual os tubulões ganham destaque por possuir uma maior capacidade de carga;
- O desenvolvimento dos códigos-fontes para execução do programa foi bem-sucedido;
- O programa foi validado com base nas exemplificações realizadas, onde o mesmo definiu adequadamente todos os parâmetros geométricos necessários para a execução de tubulões;
- O programa também executa adequadamente o cálculo das armaduras longitudinais para o arranjo escolhido no presente trabalho.

De forma geral, o trabalho apresentou o desenvolvimento e a eficiência de um programa para dimensionamento de fundações do tipo tubulões direcionadas para torres de linhas de transmissão garantindo segurança da torre a respeito de todos os esforços que a mesma se encontra submetida.

REFERÊNCIAS

ALONSO, U. R. – **Dimensionamento de fundações profundas**. 2ª. ed. Edgard Blücher Ltda. – 2012.

ALONSO, U. R. **Exercícios de Fundações**. 2ª. ed. São Paulo: Edgard Blucher Ltda, 2010.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR-5422**: Projeto de Linhas Aéreas de Transmissão de Energia Elétrica: Procedimento. Rio de Janeiro, 1985. 58. p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR-6122**: Projeto e execução de fundações. Rio de Janeiro, 1996. 22 p.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR-6118:** Projeto de estrutura de concreto: Procedimento. Rio de Janeiro, 2014. 256 p.

AMARAL, R. C. (26 de novembro de 2015). **Dimensionamento de fundações para torres metálicas de linha de transmissão de energia elétrica**. Florianópolis, Santa Catarina, Brasil.

BORGES, L. E. (2010). **Python para Desenvolvedores***.* Rio de Janeiro: Edição do Autor.

CAMPOS, J. C. D. **Elemento de fundações em concreto**. 1. ed. São Paulo: Oficina de Textos, v. Único, 2015.

CINTRA, José Carlos A.; AOKI, Nelson; ALBIERO, José Henrique. **Fundações diretas:** Projeto geotécnico. S.i: Oficina de Texto, 2011. 139 p.

CHAGAS, J. D., & OLIVEIRA, M. V. (dezembro de 2009). **Programação orientada a objetos versus programação orientada a aspectos - Um estudo de caso comparativo através do desenvolvimento de um banco de questões.** São Cristóvão, Sergipe, Brasil.

CHAVES, R. A. (30 de abril de 2004). **Fundações de Torres de Linhas de Transmissão e de Telecomunicação**. Belo Horizonte, Minas Gerais, Brasil.

COELHO, F. C. (2007). **Computação Científica com Python - Uma introdução à programação para cientistas.** (1ª ed.). Petrópolis, Rio de Janeiro: Edição do Autor.

GARCIA, O. D. (10 de Julho de 2005). **Influência da Qualidade da Compactação dos reaterros na capacidade de carga de fundações submetidas a esforços de tração.** Rio de Janeiro, Rio de Janeiro, Brasil.

GOETTEN Junior, WINCK Diogo. AspectJ – **Programação Orientada a Aspectos com Java**. São Paulo – SP: Novatec Editora, 2006.

Gontijo, C. R. – **Cálculo de torres para linhas de transmissão** – IEA Editora - 1994

LUTZ, M.; ASCHER, D. **Aprendendo python**. Tradução de João Tortello. Porto Alegre, Bookman, 2007.

QUENTAL, J. C. (24 de outubro de 2008). **Comportamento geomecânico dos solos de fundações das torres da linha de transmissão Recife II/Bongi**.

Recife, Pernambuco, Brasil. Acesso em 24 de janeiro de 2019, disponível em: https://repositorio.ufpe.br/bitstream/123456789/5315/1/arquivo2432_1.pdf.

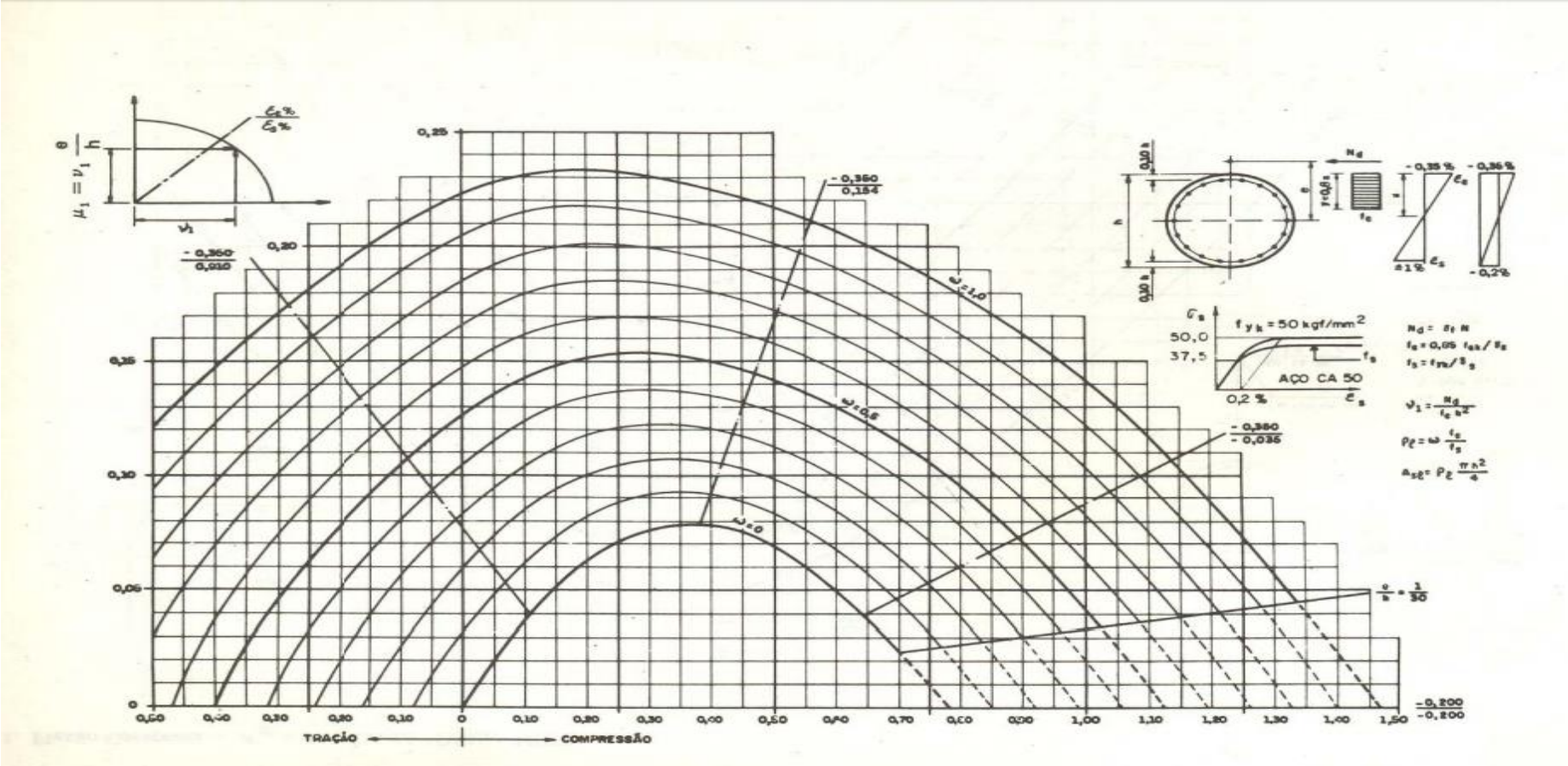
REBELLO, Y. C. (2008). **Fundações: guia prático de projeto, execução e dimensionamento***.* São Paulo: Zigurate.

SOUSA FILHO, G. F., & ALEXANDRE, E. S. (2015). **Introdução à Computação** (2ª ed., Vol. 1.1). João Pessoa: Editora da UFPB.

VELLOSO, D. D. A.; LOPES, F. D. R. **Fundações: Critérios de Projeto, Investigação do Subsolo, Fundações Superficiais, Fundações Profundas**. São Paulo: Oficina de Textos, v. Completo, 2010.

VELOZO, L. T. (27 de maio de 2010). **Metodização do estudo das fundações para suportes de linhas de transmissão**. Rio de Janeiro, Rio de Janeiro, Brasil. Acesso em 24 de janeiro de 2019, disponível em: https://www.maxwell.vrac.puc-rio.br/Busca_etds.php?strSecao=resultado&nrSeq=16178@1.

Anexo A – Ábaco 6.2 de Pfeil (1989).



Anexo B - Tabela da área de aço.

Valor nominal para cálculo		Área de aço da seção conforme número de barras – A_s [cm^2]									
ϕ diâmetro (mm)	massa linear (kg/m)	1	2	3	4	5	6	7	8	9	10
5,0	0,16	0,20	0,40	0,60	0,80	1,00	1,20	1,40	1,60	1,80	2,00
6,3	0,25	0,315	0,63	0,945	1,26	1,575	1,89	2,205	2,52	2,835	3,15
8,0	0,40	0,50	1,00	1,50	2,00	2,50	3,00	3,50	4,00	4,50	5,00
10,0	0,63	0,80	1,60	2,40	3,20	4,00	4,80	5,60	6,40	7,20	8,80
12,5	1,00	1,25	2,50	3,75	5,00	6,25	7,50	8,75	10,00	11,25	12,50
16,0	1,60	2,00	4,00	6,00	8,00	10,00	12,00	14,00	16,00	18,00	20,00
20,0	2,50	3,15	6,30	9,45	12,60	15,75	18,90	22,05	25,20	28,35	31,50
25,0	4,00	5,00	10,00	15,00	20,00	25,00	30,00	35,00	40,00	45,00	50,00
32,0	6,30	8,00	16,00	24,00	32,00	40,00	48,00	56,00	64,00	72,00	80,00
40,0	10,00	12,50	25,00	37,50	50,00	65,50	75,00	87,50	100,00	112,50	125,00

Anexo C – Código-fonte da classe carregamento.

```
# -*- coding: latin1 -*-
'''
Created on 28 de fev de 2019

@author: Renata
'''


class Carregamento (object):
    '''
    Classe com a estrutura de dados do objeto do tipo Carregamento.
    '''
    def __init__(self, fv= 0.0, ft= 0.0, fl= 0.0):
        '''
        Inicializar um objeto do tipo Carregamento.
        '''
        self._fv = fv
        self._ft = ft
        self._fl = fl
        self.calcular_fh()

    def __repr__(self):
        '''
        Representação em forma de texto do objeto do tipo Carregamento.
        '''
        return "Carregamento(fv = %.2f, ft = %.2f, fl = %.2f, fh = %.2f)"
% (self.fv,self.ft,self.fl,self.fh)

    def __str__(self):
        '''
        Representação em forma de texto do objeto do tipo Carregamento.
        '''
        return "Carregamento(fv = %.2f, ft = %.2f, fl = %.2f, fh = %.2f)"
% (self.fv,self.ft,self.fl,self.fh)

    def obter_ft (self):
        '''
        Obter o valor do atributo ft do objeto Carregamento.
        '''
        return self._ft

    def atribuir_ft (self, ft):
        '''
        Atribuir o valor do atributo ft do objeto Carregamento.
        '''
        self._ft = ft
        self.calcular_fh()
```

```
# Propriedade ft
ft = property (obter_ft, atribuir_ft)

def obter_fv (self):
    '''
    Obter o valor do atributo fv do objeto Carregamento.
    '''
    return self._fv

def atribuir_fv (self, fv):
    '''
    Atribuir o valor do atributo fv do objeto Carregamento.
    '''
    self._fv = fv
# Propriedade fv
fv = property (obter_fv, atribuir_fv)

def obter_fl (self):
    '''
    Obter o valor do atributo fl do objeto Carregamento.
    '''
    return self._fl

def atribuir_fl (self, fl):
    '''
    Atribuir o valor do atributo fl do objeto Carregamento.
    '''
    self._fl = fl
    self.calcular_fh()
# Propriedade fl
fl = property (obter_fl, atribuir_fl)

def calcular_fh(self):
    '''
    Cálculo da força horizontal.
    '''
    ft = self.ft
    fl = self.fl
    self._fh = ((ft**2)+(fl**2))**0.5

def obter_fh (self):
    '''
    Obter o valor do atributo fh do objeto Carregamento.
    '''
    self.calcular_fh()
    return self._fh

# Propriedade fh
fh = property (obter_fh)
```

Anexo D - Código-fonte da classe concreto.

```
# -*- coding: latin1 -*-
'''
Created on 20 de fev de 2019

@author: Renata
'''

class Concreto(object):
    '''
    Classe com a estrutura de dados do objeto do tipo Concreto.
    '''

    def __init__(self, fck = 0.0, gama_c = 1.6, gama_f = 1.4):
        '''
        Inicializar um objeto do tipo Concreto.
        '''
        self._fck = fck
        self._gama_c = gama_c
        self._gama_f = gama_f
        self.calcular_tensao_adm()

    def __repr__(self):
        '''
        Representação em forma de texto do objeto do tipo Concreto.
        '''
        return "Concreto(fck = %.2f, gama_c = %.2f, gama_f = %.2f)" %
(self._fck, self._gama_c, self._gama_f)

    def __str__(self):
        '''
        Representação em forma de texto do objeto do tipo Concreto.
        '''
        return "Concreto(fck = %.2f, gama_c = %.2f, gama_f = %.2f)" %
(self._fck, self._gama_c, self._gama_f)

    def obter_fck(self):
        '''
        Obter o valor do atributo fck do objeto Concreto.
        '''
        return self._fck

    def atribuir_fck (self, fck):
        '''
        Atribuir o valor do atributo fck do objeto Concreto.
        '''
        self._fck = fck
```

```
    # Propriedade fck
    fck = property (obter_fck, atribuir_fck)

    def obter_gama_c (self):
        '''
        Obter o valor do atributo gama_c do objeto Concreto.
        '''
        return self._gama_c

    def atribuir_gama_c (self, gama_c):
        '''
        Atribuir o valor do atributo gama_c do objeto Concreto.
        '''
        self._gama_c = gama_c

    # Propriedade gama_c
    gama_c = property (obter_gama_c, atribuir_gama_c)

    def obter_gama_f (self):
        '''
        Obter o valor do atributo gama_f do objeto Concreto.
        '''
        return self._gama_f

    def atribuir_gama_f (self, gama_f):
        '''
        Atribuir o valor do atributo gama_f do objeto Concreto.
        '''
        self._gama_f = gama_f

    # Propriedade gama_f
    gama_f = property (obter_gama_f, atribuir_gama_f)

    def obter_sigma_adm (self):
        '''
        Obter o valor do atributo sigma_adm do objeto Concreto.
        '''
        return self._sigma_adm

    def calcular_tensao_adm(self):
        '''
        Cálculo da tensão admissível do concreto
        '''
        self._sigma_adm = ((0.85 *
self.fck)/(self.gama_c*self.gama_f))/10


    # Propriedade sigma_adm
    sigma_adm = property (obter_sigma_adm)
```

Anexo E - Código-fonte da classe Solo.

```
# -*- coding: latin1 -*-
'''
Created on 24 de fev de 2019

@author: Renata
'''
import math
import numpy

#from Classe_Tubulao import Tubulao

class Solo (object):

    '''
    Classe com a estrutura de dados do objeto do tipo Solo.
    '''

    def __init__(self, nspt= 0.0,c= 0.0,phi= 0.0, gama_solo = 0.0, sy=
0.60, t = "areia"):
        '''
        Inicializar um objeto do tipo Solo.
        '''
        self._nspt = nspt
        self._c = c
        self._phi = phi
        self._gama_solo = gama_solo
        self._sy = sy
        self.calcular_tensao_adm()
        self._tipo = t
        if t == "areia":
            self.r = 20
        elif t == "argila":
            self.r = 30

    def __repr__(self):
        '''
        Representação em forma de texto do objeto do tipo Solo.
        '''
        return "Solo(nspt = %.2f, c = %.2f, phi = %.2f, gama_solo = %.2f,
sy = %.2f)" % (self.nspt) % (self.c) % (self.phi) % (self.gama_solo) %
(self.sy)

    def obter_nspt (self):
        '''
```

```
        Obter o valor do atributo Nspt do objeto Solo.
        '''
        return self._nspt

    def atribuir_nspt (self, nspt):
        '''
        Atribuir o valor do atributo Nspt do objeto Solo.
        '''
        self._nspt = nspt
        self.calcular_tensao_adm()

    # Propriedade Nspt
    nspt = property (obter_nspt, atribuir_nspt)

    def obter_c (self):
        '''
        Obter o valor do atributo c do objeto Solo.
        '''
        return self._c

    def atribuir_c (self, c):
        '''
        Atribuir o valor do atributo c do objeto Solo.
        '''
        self._c = c

    # Propriedade c
    c = property (obter_c, atribuir_c)

    def obter_phi (self):
        '''
        Obter o valor do atributo phi do objeto Solo.
        '''
        return self._phi

    def atribuir_phi (self, phi):
        '''
        Atribuir o valor do atributo phi do objeto Solo.
        '''
        self._phi = phi

    # Propriedade phi
    phi = property (obter_phi, atribuir_phi)

    def obter_gama_solo (self):
        '''
        Obter o valor do atributo gama do solo do objeto Solo.
        '''
        return self._gama_solo
```

```
    def atribuir_gama_solo (self, gama_solo):
        '''
        Atribuir o valor do atributo gama_solo do objeto Solo.
        '''
        self._gama_solo = gama_solo

    # Propriedade gama_solo
    gama_solo = property (obter_gama_solo, atribuir_gama_solo)

    def obter_sy (self):
        '''
        Obter o valor do atributo Sy do objeto solo.
        '''
        return self._sy

    def atribuir_sy (self, sy):
        '''
        Atribuir o valor do atributo Sy do objeto solo.
        '''
        self._sy = sy

    # Propriedade sy
    sy = property(obter_sy,atribuir_sy)

    def calcular_tensao_adm(self):
        '''
        Cálculo da tensão admissível do solo
        '''
        self._sigma_adm = (self.nspt/5)*0.00981

    def obter_sigma_adm (self):
        '''
        Obter o valor do atributo sigma_adm do objeto Solo.
        '''
        return self._sigma_adm

    # Propriedade sigma_adm
    sigma_adm = property (obter_sigma_adm)

    def calcular_nq (self):
        '''
        Cálculo do fator de capacidade de carga (Nq) do solo.
        '''
        self._nq =
(numpy.tan(numpy.radians(45+(0.5*self.phi))**2))*numpy.exp(math.pi*numpy.
tan(numpy.radians(self.phi)))

    def obter_nq (self):
```

```
        '''
        Obter o valor do atributo Nq do objeto Solo.
        '''
        self.calcular_nq()
        return self._nq

    # Propriedade nq
    nq = property (obter_nq)

    def calcular_nc (self):
        '''
        Cálculo do fator de capacidade de carga (Nc) do solo.
        '''
        if numpy.tan(numpy.radians(self.phi))==0:
            self._nc = 0
        else:
            self._nc = numpy.power(numpy.tan(numpy.radians(self.phi)),-
1)*(self.nq-1)


    def obter_nc (self):
        '''
        Obter o valor do atributo Nc do objeto Solo.
        '''
        self.calcular_nc()
        return self._nc

    # Propriedade nc
    nc = property (obter_nc)

    def calcular_ny (self):
        '''
        Cálculo do fator de capacidade de carga (Ny) do solo.
        '''
        self._ny = (2*numpy.tan(numpy.radians(self.phi)))*(self.nq+1)

    def obter_ny (self):
        '''
        Obter o valor do atributo Ny do objeto Solo.
        '''
        self.calcular_ny()
        return self._ny

    # Propriedade ny
    ny = property (obter_ny)

    def calcular_sq (self):
        '''
        Cálculo do fator de forma (Sq) do solo.
```

```
    '''
    self._sq = (1+numpy.tan(numpy.radians(self.phi)))

def obter_sq (self):
    '''
    Obter o valor do atributo Sq do objeto Solo.
    '''
    self.calcular_sq()
    return self._sq

# Propriedade sq
sq = property (obter_sq)

def calcular_sc (self):
    '''
    Cálculo do fator de forma (Sc) do solo.
    '''
    if self.nc==0:
        self._sc = 0
    else:
        self._sc = (1+(self.nq/self.nc))

def obter_sc (self):
    '''
    Obter o valor do atributo Sc do objeto Solo.
    '''
    self.calcular_sc()
    return self._sc

# Propriedade sc
sc = property (obter_sc)

def obter_fadm (self):

    res = 0.0
    if self.nspt <=4:
        res = 10
    elif self.nspt>4 and self.nspt<=8:
        res = 16
    elif self.nspt >20 :
        res = 30
    self._fadm = res
    return self._fadm

# Propriedade fadm
fadm = property (obter_fadm)
```

Anexo F – Código-fonte da classe Tubulão.

```
# -*- coding: utf-8
'''
Created on 28 de fev de 2019

@author: Renata
'''
import math
import numpy

from Classe_Concreto import Concreto
from Classe_Solo import Solo
from Classe_Carregamento import Carregamento

class Tubulao(object):

    '''
    Classe com a estrutura de dados do objeto do tipo Tubulão.
    '''


    def __init__(self, cota= 0.0, g = 0.0, w = 0.0, fyk = 500.0,
carregamento = None, concreto = None, solo = None):
        '''
        Inicializar um objeto do tipo Tubulão.
        '''
        self._cota = cota
        self._g = g
        self._w = w
        self.fyk = fyk
        self._carregamento = carregamento
        self._concreto = concreto
        self._solo = solo
        self.tab_barras =
[[10.0,0.8],[12.5,1.25],[16.0,2.0],[20.0,3.15],[25.0,5.0],[32.0,8.0],[40.
0,12.5]]


    def __repr__(self):
        '''
        Representação em forma de texto do objeto do tipo Tubulão.
        '''
        return "Tubulao(cota = %.2f, g = %.2f, w = %.2f, fyk = %.2f,
carregamento = %s, concreto = %s, solo = %s)" % (self._cota, self._g,
self._w, self._fyk, self._carregamento, self._concreto, self._solo)

    def __str__(self):
        '''
        Representação em forma de texto do objeto do tipo Tubulao.
```

```
        '''
        return "Tubulao(cota = %.2f, g = %.2f, w = %.2f, fyk = %.2f,
carregamento = %s, concreto = %s, solo = %s)" % (self._cota, self._g,
self._w, self._fyk, self._carregamento, self._concreto, self._solo)

    def obter_cota (self):
        '''
        Obter o valor do atributo cota do objeto Tubulao.
        '''
        return self._cota

    def atribuir_cota (self, cota):
        '''
        Atribuir o valor do atributo cota do objeto Tubulão.
        '''
        self._cota = cota

    # Propriedade cota
    cota = property (obter_cota, atribuir_cota)

    def obter_g (self):
        '''
        Obter o valor do atributo g do objeto Tubulao.
        '''
        return self._g

    def atribuir_g (self, g):
        '''
        Atribuir o valor do atributo g do objeto Tubulão.
        '''
        self._g = g

    # Propriedade g
    g = property (obter_g, atribuir_g)

    def obter_w (self):
        '''
        Obter o valor do atributo w do objeto Tubulao.
        '''
        return self._w

    def atribuir_w (self, w):
        ''''
        Atribuir o valor do atributo w do objeto Tubulao.
        '''
        self._w = w

    #Propriedade w
    w = property (obter_w, atribuir_w)
```

```
def obter_fyk (self):
    '''
    Obter o valor característico do aço.
    '''
    return self._fyk

def atribuir_fyk (self, fyk):
    '''
    Atribuir o valor do atributo fyk do objeto Tubulão.
    '''
    self._fyk = fyk

#Propriedade fyk
fyk = property (obter_fyk, atribuir_fyk)



def obter_carregamento (self):
    '''
    Obter o valor do atributo carregamento do objeto Tubulão.
    '''
    return self._carregamento

def atribuir_carregamento (self, carregamento):
    '''
    Atribuir o valor do atributo carregamento do objeto Tubulão.
    '''
    self._carregamento = carregamento

# Propriedade carregamento
carregamento = property (obter_carregamento, atribuir_carregamento)

def obter_concreto (self):
    '''
    Obter o valor do atributo concreto do objeto Tubulão.
    '''
    return self._concreto

def atribuir_concreto (self, concreto):
    '''
    Atribuir o valor do atributo concreto do objeto Tubulão.
    '''
    self._concreto = concreto

# Propriedade concreto
concreto = property (obter_concreto, atribuir_concreto)

def obter_solo (self):
```

```
        '''
        Obter o valor do atributo solo do objeto Tubulao.
        '''
        return self._solo

    def atribuir_solo (self, solo):
        '''
        Atribuir o valor do atributo solo do objeto Tubulao.
        '''
        self._solo = solo

    # Propriedade solo
    solo = property (obter_solo, atribuir_solo)

    def calcular_db (self):
        '''
        Calcular o valor do diâmetro da base.
        '''
        fv = self.carregamento.fv
        sigma_adm = self.solo.sigma_adm
        self._db =  ((4*fv/(sigma_adm*math.pi))**0.5)

    def obter_db (self):
        '''
        Obter o valor do atributo db do objeto Tubulao.
        '''
        self.calcular_db()
        return self._db

    # Propriedade db
    db = property (obter_db)

    def calcular_df (self):
        '''
        Calcular o valor do diâmetro do fuste.
        '''
        fv = self.carregamento.fv
        sigma_adm = self.concreto.sigma_adm

        self._df = ((4*fv/(sigma_adm*math.pi))**0.5)
        if self._df<70:
            self._df = 70

    def obter_df (self):
        '''
        Obter o valor do atributo df do objeto Tubulao.
        '''
        self.calcular_df()
        return self._df
```

```
# Propriedade df
df = property (obter_df)

def calcular_ab (self):
    '''
    Calcular o valor da área da base do tubulão.
    '''
    self._ab = ((math.pi*(self.db**2))/4)

def obter_ab (self):
    '''
    Obter o valor do atributo ab do objeto Tubulao.
    '''
    self.calcular_ab()
    return self._ab

# Propriedade ab
ab = property (obter_ab)

def calcular_af (self):
    '''
    Calcular o valor da área do fuste do tubulão.
    '''
    self._af = ((math.pi*(self.df**2))/4)

def obter_af (self):
    '''
    Obter o valor do atributo af do objeto Tubulao.
    '''
    self.calcular_af()
    return self._af

# Propriedade af
af = property (obter_af)

def calcular_hc (self):
    '''
    Calcular o valor da altura do cone.
    '''
    self._hc = ((self.db-self.df)*0.866)
    if self._hc > 200:
        self._hc = 200

def obter_hc (self):
    '''
    Obter o valor do atributo hc do objeto Tubulao.
```

```
        '''
        self.calcular_hc()
        return self._hc

    # Propriedade hc
    hc = property (obter_hc)

    def calcular_hf (self):
        '''
        Calcular o valor da altura do fuste.
        '''
        self._hf = (self.cota-self.hc)

    def obter_hf (self):
        '''
        Obter o valor do atributo hf do objeto Tubulao.
        '''
        self.calcular_hf()
        return self._hf

    # Propriedade hf
    hf = property (obter_hf)

    def calcular_vb (self):
        '''
        Calcular o valor do volume da base.
        '''

        self._vb = ((0.2*self.ab)+((self.hc-
20)/3)*(self.ab+self.af+((self.ab*self.af)**0.5)))

    def obter_vb (self):
        '''
        Obter o valor do atributo vb do objeto Tubulao.
        '''
        self.calcular_vb()
        return self._vb

    # Propriedade vb
    vb = property (obter_vb)

    def calcular_vf (self):
        '''
        Calcular o valor do volume do fuste.
        '''
        self._vf = (math.pi*(self.df/2)**2*self.hf)

    def obter_vf (self):
        '''
```

```
        Obter o valor do atributo vf do objeto Tubulao.
        '''
        self.calcular_vf()
        return self._vf

    # Propriedade vf
    vf = property (obter_vf)

    def calcular_vt (self):
        '''
        Calcular o valor do volume total do tubulao.
        '''
        self._vt = (self.vb+self.vf)

    def obter_vt (self):
        '''
        Obter o valor do atributo vt do objeto Tubulao.
        '''
        self.calcular_vt()
        return self._vt

    # Propriedade vt
    vt = property (obter_vt)

    def calcular_asm (self):
        '''
        Calcular o valor da área de aço mínima.
        '''
        self._asm = (0.005*math.pi*(self.df**2))/4

    def obter_asm (self):
        '''
        Obter o valor do atributo as do objeto Tubulao.
        '''
        self.calcular_asm()
        return self._asm

    # Propriedade asm
    asm = property (obter_asm)

    def calcular_tensao_ult (self):
        '''
        Calcular o valor da tensão última do solo onde o tubulao está
assentado.
        '''
        c = self.solo.c
        nc = self.solo.nc
        sc = self.solo.sc
        gama_solo = self.solo.gama_solo
```

```
        ny = self.solo.ny
        sy = self.solo.sy
        nq = self.solo.nq
        sq = self.solo.sq

        self._tensao_ult =
(c*nc*sc)+(0.5*gama_solo*(self.db/100)*ny*sy)+(gama_solo*(self.cota/100)*
nq*sq)

    def obter_tensao_ult (self):
        '''
        Obter o valor do atributo da tensao ultima do objeto Tubulao.
        '''
        self.calcular_tensao_ult()
        return self._tensao_ult

    # Propriedade tensao_ult
    tensao_ult = property (obter_tensao_ult)

    def calcular_tensao_base (self):
        '''
        Calcular a tensão do solo na base do tubulao.
        '''
        fv = self.carregamento.fv
        fadm = self.solo.fadm

        self._tensao_base = (fv+(2500*self.vt*9.81*0.001)-
(math.pi*(self.df/100)*((self.hf-self.db)/100))*fadm)/(self.ab*0.0001)

    def obter_tensao_base (self):
        '''
        Obter o valor do atributo da tensao base do objeto Tubulao.
        '''
        self.calcular_tensao_base()
        return self._tensao_base

    # Propriedade tensao_base
    tensao_base = property (obter_tensao_base)


    def calcular_d (self):
        '''
        Calcular o diametro maior do tronco de cone.
        '''
        r = self.solo.r
        self._d = self.db + (2*self.hf*(numpy.tan(numpy.radians(r))))

    def obter_d (self):
        '''
```

```
        Obter o valor do atributo d.
        '''
        self.calcular_d()
        return self._d

    # Propriedade _d
    d = property (obter_d)

    def calcular_ve (self):
        '''
        Calcular o volume de terra no método do cone.
        '''
        self._ve =
(((math.pi*self.hf)/12)*(self.d**2*+self.d*self.db+self.db**2))-self.vt

    def obter_ve (self):
        '''
        Obter o valor do atributo ve.
        '''
        self.calcular_ve()
        return self._ve

    # Propriedade ve
    ve = property (obter_ve)

    def calcular_pu (self):
        '''
        Calcular o valor do equilibrio vertical.
        '''
        self._pu = (2500*self.vt*9.81*0.001)+(2650*self.ve*9.81*0.001)

    def obter_pu (self):
        '''
        Obter o valor do atributo pu.
        '''
        self.calcular_pu()
        return self._pu

    # Propriedade pu
    pu = property (obter_pu)

    def verificacao_compressao (self):
        self.obter_tensao_base()
        self.obter_tensao_ult()
        '''
        Cálculo da verificação à compressão.
        '''
        self._vco = "vco"
        if self.tensao_base <= (self.tensao_ult/3):
```

```
            self._vco = "ok"
        elif self.tensao_base > (self.tensao_ult/3):
            self._vco = "não ok"
        print("verificação a compressão: %s"%self._vco)
        #self.vco = "verificacao a compressao"

    def verificacao_arrancamento (self):
        self.obter_pu()
        '''
        Verificacao ao arrancamento do tubulao.
        '''
        fv = self.carregamento.fv

        self._vaa = "vaa"
        if self._pu>= (fv*1.10):
            self._vaa = "ok"
        elif self._pu< (fv*1.10):
            self._vaa = "não ok"
        print("verificação ao arrancamento: %s"%self._vaa)
        #self.vaa = "verificacao ao arrancamento"

    def calcular_v (self):
        '''
        Calcular o coeficiente v do aço.
        '''
        fh = self.carregamento.fh
        gama_f = self.concreto.gama_f
        fck = self.concreto.fck
        self._v = ((fh*gama_f)/((fck/10)*self.db**2))

    def obter_v (self):
        '''
        Obter o valor do atributo v.
        '''
        self.calcular_v()
        return self._v

    # Propriedade v
    v = property (obter_v)


    def calcular_md (self):
        '''
        Calcular o valor do momento de cálculo.
        '''
        fh = self.carregamento.fh
        gama_solo = self.solo.gama_solo
        phi = self.solo.phi
```

```
        self._md = fh*(self.g + 0.545*(fh/(gama_solo +
self.df*(numpy.tan(numpy.radians(45+(phi/2)))**2)))**0.5)

    def obter_md (self):
        '''
        Obter o valor do atributo md.
        '''
        self.calcular_md()
        return self._md

    # Propriedade md
    md = property (obter_md)

    def calcular_e (self):
        '''
        Calcular o valor da excentricidade.
        '''
        fh = self.carregamento.fh
        gama_f = self.concreto.gama_f

        self._e = (self.md/(fh*gama_f))

    def obter_e (self):
        '''
        Obter o valor do atributo e.
        '''
        self.calcular_e()
        return self._e

    # Propriedade e
    e = property (obter_e)

    def calcular_m (self):
        '''
        Calcular o valor do coeficiente m.
        '''
        self._m = (self.v*(self.e/(self.df/100)))

    def obter_m (self):
        '''
        Obter o valor do atributo m.
        '''
        self.calcular_m()
        return self._m

    # Propriedade m
    m = property (obter_m)

    def calcular_p (self):
```

```
        '''
        Calcular o valor do coeficiente p (rô).
        '''
        fck = self.concreto.fck

        self._p = (self.w*((fck/10)/(self.fyk/10)))

    def obter_p (self):
        '''
        Obter o valor do atributo p.
        '''
        if self.w == 0.0:
            self.w = float(input("Entre com o valor da taxa de aco: "))

        self.calcular_p()
        return self._p

    #Propriedade p
    p = property (obter_p)

    def calcular_ast (self):
        '''
        Calcular a área de aço necessária para o tubulão.
        '''
        self._ast = (self.p*self.af)
        if self._ast < self.asm :
            self._ast = self.asm

    def obter_ast (self):
        '''
        Obter o valor da área de aço.
        '''
        self.calcular_ast()
        return self._ast

    #Propriedade ast
    ast = property (obter_ast)

    def calcular_asu (self):
        '''
        Calcular a área de aço unitária.
        '''
        self._asu = (self.ast/18)

    def obter_asu (self):
        '''
        Obter o valor da área de aço unitária.
        '''
        self.calcular_asu()
```

```
        return self._asu

    #Propriedade asu
    asu = property (obter_asu)

    def obter_diametro_barra (self):
        self.d_barra = 0
        '''
        Obter o valor do diâmetro da barra do arranjo.
        '''
        asu = self.asu
        for barra in self.tab_barras:
            if asu < barra[1]:
                self.d_barra = barra[0]
                break
        return self.d_barra

    #Propriedade diametro_barra
    diametro_barra = property (obter_diametro_barra)


    def calcular_vrd (self):
        '''
        Cálculo da resistência à compressão diagonal da armadura.
        '''
        fck = self.concreto.fck
        self._vrd = 0.27*(1-
((fck/10)/250))*((fck/10)/1.4)*self.df*((self.df)-
((self.d_barra/10)/2+(0.1*self.df)))

    def obter_vrd (self):
        '''
        Obter o valor da resistência à compressão diagonal da armadura
        '''
        self.calcular_vrd()
        return self._vrd

    #Propriedade vrd
    vrd = property (obter_vrd)

    def verificacao_comp_diag (self):
        self.obter_vrd()
        '''
        Cálculo da verificação à compressão diagonal.
        '''
        fv = self.carregamento.fv
        self.vcd = "vcd"
        if self._vrd >= fv:
            self.vcd = "ok"
```

```
        elif self._vrd < fv:
            self.vcd = "Não ok"
        print("verificação a compressão diagonal: %s"%self.vcd)

    def armadura_transv (self):
        self.obter_diametro_barra()
        '''
        Definir a armadura transversal.
        '''
        self.d_transv = 0
        if (self.d_barra)/4>5:
            self.d_transv = self.d_barra/4
        else:
            self.d_transv = 5
        return self.d_transv

    def espacamento_w(self):
        '''
        Definir espaçamento da armadura transversal
        '''
        self.obter_diametro_barra()
        esp = 24*self.d_barra/10
        self.sw = 0
        if 20<self.df and 20<esp:
            self.sw = float(20)
        elif self.df<20 and self.df<esp:
            self.sw = float(self.df)
        else:
            self.sw = float(esp)

        return self.sw

    def calcular_vc_1 (self):
        '''
        Cálculo da resistência na flexão simples e na flexo-tração.
        '''
        fck = self.concreto.fck
        self._vc_1 = (0.6*((0.21*(fck/10)**(2/3))/1.4)*self.df*(self.df-
(0.1*self.df)-(self.d_barra/2)))
        #self.vc_1 =
self.df#(0.6*((0.21*(fck/10)**(2/3))/1.4)*self.df*(self.df-(0.1*self.df)-
(self.d_barra/2)))

    def obter_vc_1 (self):
        '''
        Obter o valor da resistência na flexão simples e na flexo-tração.
        '''
        self.calcular_vc_1()
        return self._vc_1
```

```
    #Propriedade vc_1
    vc_1 = property (obter_vc_1)

    def calcular_vsw (self):
        self.espacamento_w()
        '''
        Cálculo da resitência a armadura transversal.
        '''
        self._vsw =
((self.hf/self.sw)*(math.pi*(self.df**2)/4)/self.sw)*(0.9*(self.df-
(0.1*self.df)-(self.d_barra/2))*((self.fyk/10)/1.15))

    def obter_vsw (self):
        '''
        Obter o valor da resistência a armadura transversal.
        '''
        self.calcular_vsw()
        return self._vsw

    #Propriedade vsw
    vsw = property (obter_vsw)

    def verificacao_armad_trans (self):
        self.obter_vc_1()
        self.obter_vsw()
        '''
        Verificação a resistência da armadura transversal.
        '''
        fh = self.carregamento.fh
        self.vat = "vat"

        if fh <= (self.vc_1 + self.vsw):
            self.vat = "ok"
        else:
            self.vat = "não ok"
        print("verificação a armadura transversal: %s"%self.vat)


    def processar(self):
        self.calcular_db()
        self.calcular_df()
        self.calcular_hc()
        self.calcular_hf()
        self.calcular_ab()
        self.calcular_af()
        self.calcular_vb()
        self.calcular_vf()
        self.calcular_vt()
```

```
        self.calcular_asm()
        self.calcular_ast()
        self.verificacao_compressao()
        self.verificacao_arrancamento()
        self.verificacao_comp_diag()
```

Anexo G – Código-fonte da classe drawTubular.

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objs as go

import numpy as np

class tubular():
    #Todas as unidades estão dadas em cm
    def __init__(self, alturaCone, alturaTotal, dF, dB):
        self.alturaCone = alturaCone
        self.alturaTotal = alturaTotal
        self.dF = dF#Diâmetro do fuste
        self.dB = dB#Diâmetro da base
        self.lb = 20#largura do rodapé
        self.la = alturaCone-self.lb
        self.lf = alturaTotal - alturaCone
        self.shapes = []#Os desenhos que foram feitos
        self.app = dash.Dash()
        self.xRange = [0,self.alturaTotal*1.1+1.6*self.dB]
        self.yRange = (-1.6*self.dB,self.alturaTotal*1.3)
        self.figureSizeX = self.xRange[0]+self.xRange[1]
        self.figureSizeY = np.abs(self.yRange[0])+self.yRange[1]

    def desenharTubular(self,pos,**kwargs):
        tubular = {
            'type':'path',
            'path' : 'M {},{} L{},{} L{},{} L{},{} L{},{} L{},{} L{},{}
L{},{} Z'.format(
                pos[0],pos[1],
                pos[0],pos[1]+self.lb,
                pos[0]+(self.dB/2)-self.dF/2,pos[1]+self.alturaCone,
                pos[0]+(self.dB/2)-self.dF/2,pos[1]+self.alturaTotal,
                pos[0]+(self.dB/2)+self.dF/2,pos[1]+self.alturaTotal,
                pos[0]+(self.dB/2)+self.dF/2,pos[1]+self.alturaCone,
                pos[0]+self.dB,pos[1]+self.lb,
                pos[0]+self.dB,pos[1],
            ),
        }
        for k,v in kwargs.items():
```

```
        tubular[k] = v
    self.shapes.append(tubular)
    #linhas horizontais para tubular
    self.desenharLinha(
        pos[0]+(self.dB/2)+self.dF/2,pos[1]+self.alturaTotal,
        3*(pos[0]+(self.dB/2)+self.dF/2),pos[1]+self.alturaTotal,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        pos[0]+(self.dB/2)+self.dF/2,pos[1]+self.la+self.lb,
        2*(pos[0]+(self.dB/2)+self.dF/2),pos[1]+self.la+self.lb,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        pos[0]+self.dB,pos[1],
        3*(pos[0]+(self.dB/2)+self.dF/2),pos[1],
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    #linhas verticais para tubular

    self.desenharLinha(
        3*(pos[0]+(self.dB/2)+self.dF/2),pos[1],
        3*(pos[0]+(self.dB/2)+self.dF/2),pos[1]+self.alturaTotal,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        2*(pos[0]+(self.dB/2)+self.dF/2),pos[1]+self.la+self.lb,
        2*(pos[0]+(self.dB/2)+self.dF/2),pos[1]+self.alturaTotal,
        line= {
```

```
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    #linhas verticais para circulos
    self.desenharLinha(
        pos[0],pos[1],
        pos[0],-self.dB,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        pos[0]+self.dB,pos[1],
        pos[0]+self.dB,-self.dB,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        self.dB/2+self.dF/2,pos[1],
        self.dB/2+self.dF/2,-self.dB,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        self.dB/2-self.dF/2,pos[1],
        self.dB/2-self.dF/2,-self.dB,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        2*(pos[0]+(self.dB/2)+self.dF/2),-self.dB+self.dF/2,
```

```
        2*(pos[0]+(self.dB/2)+self.dF/2),-self.dB-self.dF/2,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        3*(pos[0]+(self.dB/2)+self.dF/2),-self.dB+self.dB/2,
        3*(pos[0]+(self.dB/2)+self.dF/2),-self.dB-self.dB/2,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    #Linhas horizontais para circulo

    self.desenharLinha(
        self.dB/2,-self.dB/2,
        3*(pos[0]+(self.dB/2)+self.dF/2),-self.dB/2,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        self.dB/2,-1.5*self.dB,
        3*(pos[0]+(self.dB/2)+self.dF/2),-1.5*self.dB,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )

    self.desenharLinha(
        self.dB/2,-self.dB+self.dF/2,
        2*(pos[0]+(self.dB/2)+self.dF/2),-self.dB+self.dF/2,
        line= {
            'color': 'rgb(255, 140, 184)',
            'width': 2,
            'dash': 'dot',
        },
    )
```

```
        self.desenharLinha(
            self.dB/2,-self.dB-self.dF/2,
            2*(pos[0]+(self.dB/2)+self.dF/2),-self.dB-self.dF/2,
            line= {
                'color': 'rgb(255, 140, 184)',
                'width': 2,
                'dash': 'dot',
            },
        )

        self.pontos = []
        p1 = go.Scatter(
            x = [
                    2*(pos[0]+(self.dB/2)+self.dF/2),
                    3*(pos[0]+(self.dB/2)+self.dF/2),
                    3*(pos[0]+(self.dB/2)+self.dF/2),
                    2*(pos[0]+(self.dB/2)+self.dF/2),
                ],
            y = [
                    self.lb+self.la+self.lf/2,
                    self.alturaTotal/2,
                    -self.dB/2,
                    -self.dB-self.dF/2,
                ],

            mode = 'text',
            name = 'Text',
            text = [
                ' hf = {} cm'.format(np.around(self.lf,3)),
                ' ht = {} cm'.format(np.around(self.alturaTotal,3)),
                ' db = {} cm'.format(np.around(self.dB,3)),
                ' df = {} cm'.format(np.around(self.dF,3)),
                ],
            orientation='v',
            textposition = 'middle right'
        )

        self.pontos.append(p1)

    def desenharArmadura(self,x0,y0,fEscala):
        angulo = np.linspace(0,2*np.pi,num=18)
        x = fEscala*.5*self.dF*np.cos(angulo) + x0
        y = fEscala*.5*self.dF*np.sin(angulo) + y0

        pontos = go.Scatter(
            x = x,
            y = y,
            mode = 'markers',
```

```
            marker = dict(
                size = 6,
                color = 'rgb(255, 140, 184)',
            ),
            text=None,
        )
        self.pontos.append(pontos)

        self.desenharCirculo([x0,y0],fEscala*self.dF/2,
        fillcolor= 'rgba(255, 140, 184, 0)',
            line = {
                        'color': 'rgb(255, 140, 184)',
                        'width':1,
                     },  )

        self.desenharCirculo([x0,y0],fEscala*1.1*self.dF/2,
        fillcolor= 'rgba(255, 140, 184, 0)',
            line = {
                        'color': 'rgb(255, 140, 184)',
                        'width':1,
                     },  )

        l = {
                'color': 'rgb(255, 140, 184)',
                'width': 2,
                'dash': 'dot',
            }
        #Interna
        self.desenharLinha(
            x[9],y[9],x[9]-fEscala*self.dF/2,y[9],
            line= l,
        )

        self.desenharLinha(
            x[8],y[8],x[9]-fEscala*self.dF/2,y[8],
            line= l,
        )
        self.desenharLinha(
            x[9]-fEscala*self.dF/2,y[8],x[9]-fEscala*self.dF/2,y[9],
            line= l,
        )
        #Externa
        self.desenharLinha(
            fEscala*(1.1*.5*self.dF*np.cos(np.pi/2))+ x0,
fEscala*(1.1*.5*self.dF*np.sin(np.pi/2)) + y0,
            -2*fEscala*self.dF+fEscala*(1.1*.5*self.dF*np.cos(np.pi/2)) +
x0, fEscala*(1.1*.5*self.dF*np.sin(np.pi/2)) + y0,
            line=l,
        )
```

```
        self.desenharLinha(
            fEscala*(1.1*.5*self.dF*np.cos(np.pi*3/2))+ x0,
fEscala*(1.1*.5*self.dF*np.sin(np.pi*3/2)) + y0,
            -2*fEscala*self.dF+fEscala*(1.1*.5*self.dF*np.cos(np.pi*3/2))
+ x0, fEscala*(1.1*.5*self.dF*np.sin(np.pi*3/2)) + y0,
            line=l,
        )

        self.desenharLinha(
            -2*fEscala*self.dF+fEscala*(1.1*.5*self.dF*np.cos(np.pi*3/2))
+ x0, fEscala*(1.1*.5*self.dF*np.sin(np.pi/2)) + y0,
            -2*fEscala*self.dF+fEscala*(1.1*.5*self.dF*np.cos(np.pi*3/2))
+ x0, fEscala*(1.1*.5*self.dF*np.sin(np.pi*3/2)) + y0,
            line=l,
        )

        #cotas

        pontos = go.Scatter(
            x = [
                    x[9]-2.7*fEscala*self.dF/2,
                    -
2.7*fEscala*self.dF+fEscala*(1.1*.5*self.dF*np.cos(np.pi*3/2)) + x0,
                ],
            y = [
                    y[9],
                    fEscala*(1.1*.5*self.dF*np.sin(np.pi/2)) + y0,
                ],

            mode = 'text',
            name = 'Text',
            text = [
                ' s = {} cm'.format(np.around(np.abs(y[8]-
y[9])/fEscala,3)),
                ' df = {} cm'.format(np.around(self.dF,3))
                ],
            orientation='v',
            textposition = 'middle right'
        )

        self.pontos.append(pontos)

    def desenharFuro(self,pos,**kwargs):
        shape = {
            'type':'path',
            'path':'M {},{} L{},{} L{},{} L{},{} Z'.format(
                pos[0]+self.dB/2-self.dF/2,pos[1],
                pos[0]+self.dB/2-self.dF/2,pos[1]+self.alturaTotal,
```

```
                pos[0]+self.dB/2+self.dF/2,pos[1]+self.alturaTotal,
                pos[0]+self.dB/2+self.dF/2,pos[1],
            )

        }
        for k,v in kwargs.items():
            shape[k] = v
        self.shapes.append(shape)

    def  desenharCirculo(self,pos,r,**kwargs):
        shape = {
            'type':'circle',
            'xref':'x',
            'yref':'y',
            'x0': pos[0]-r,
            'y0': pos[1]-r,
            'x1': pos[0]+r,
            'y1': pos[1]+r,
        }
        for k,v in kwargs.items():
            shape[k] = v
        self.shapes.append(shape)

    def desenharLinha(self, x0, y0, x1, y1,**kwargs):
        shape = {
            'type':'line',
            'x0':x0,
            'y0':y0,
            'x1':x1,
            'y1':y1,
        }
        for k,v in kwargs.items():
            shape[k] = v
        self.shapes.append(shape)


    def montarLayout(self):
        self.layout = {
            'showlegend':False,
            'xaxis': {
                'range': self.xRange,
                'zeroline': False,
                'autorange':True,
                'showgrid':False,
                'showline':False,
                'ticks':'',
                'showticklabels':False,
                'autosize':False,
                'domain':[0, 1],
```

```
        },
        'yaxis': {
            'range': self.yRange,
            'zeroline': False,
            'autorange':True,
            'showgrid':False,
            'showline':False,
            'ticks':'',
            'showticklabels':False,
            'autosize':False,
            'scaleanchor':"x",
            'domain':[0, 1],
        },
        'shapes':self.shapes,
        'height':600,
        'width':800,
        'autorange':False,
        'autosize':False,
        "dimension": "ratio",
        'aspectratio':dict(x=1,y=1),
        'aspectmode':'data'

    }

def montarHTML(self):
    self.app.layout = html.Div(children=[
        dcc.Graph(
            id='dimensoes',
            figure={
                'data':self.pontos,
                'layout':self.layout,
            }
        )
    ])


def desenhar(self):
    self.desenharTubular([0,0],
    fillcolor='rgba(255, 140, 184, 0.5)',
    line={
            'color': 'rgb(255, 140, 184)',
            'width':.8,
        },
        )
    self.desenharFuro([0,0],
        fillcolor= 'rgba(255, 140, 184, 0.6)',
        line = {
                    'color': 'rgb(255, 140, 184)',
                    'width':.9
```

```
                },
        )

        self.desenharCirculo([self.dB/2,-self.dB],self.dB/2,
            fillcolor='rgba(255, 140, 184, 0.5)',
                line={
                        'color': 'rgb(255, 140, 184)',
                        'width':.8,
                },
        )

        self.desenharCirculo([self.dB/2,-self.dB],self.dF/2,
            fillcolor= 'rgba(255, 140, 184, 0.6)',
            line = {
                        'color': 'rgb(255, 140, 184)',
                        'width':.9
                },
        )


        self.desenharArmadura(10*self.dB,self.alturaTotal-self.dB,4)
        self.montarLayout()
        self.montarHTML()
        self.app.run_server(debug=True)


#t = tubular(1.1, 8, 1.5, 2.5)
#t.desenhar()
```