UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA CIVIL

LUIZ AUGUSTO DIAMANTE RIBEIRO

OTIMIZAÇÃO ESTRUTURAL DE TRELIÇAS UTILIZANDO O ALGORITMO FIREFLY

Trabalho de conclusão de curso submetido ao Curso de Graduação da Universidade Federal de Santa Catarina para a obtenção de Grau de Engenheiro Civil.

Orientador: Prof. Rafael Holdorf Lopez, Dr.

FLORIANÓPOLIS
2014

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Ribeiro, Luiz Augusto Diamante
OTIMIZAÇÃO ESTRUTURAL DE TRELIÇAS UTILIZANDO O ALGORITMO FIREFLY / Luiz Augusto Diamante Ribeiro ; orientador, Rafael Holdorf Lopez - Florianópolis, SC, 2014.
113 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico. Graduação em Engenharia Civil.

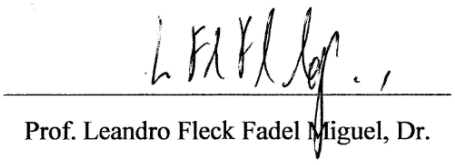
Inclui referências

1. Engenharia Civil. 2. Otimização estrutural. 3. Algorimo Firefly. 4. Algoritmo Metaheurístico. 5. Treliças. I. Lopez, Rafael Holdorf. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Civil. III. Título.

LUIZ AUGUSTO DIAMANTE RIBEIRO

OTIMIZAÇÃO ESTRUTURAL DE TRELIÇAS UTILIZANDO O ALGORITMO FIREFLY

Trabalho de Conclusão de Curso aprovado como requisito parcial para a obtenção de grau de Engenheiro Civil junto ao Departamento de Engenharia Civil do Centro Tecnológico da Universidade Federal de Santa Catarina.



Prof. Leandro Fleck Fadel Miguel, Dr.

Universidade Federal de Santa Catarina

Membro da Banca Examinadora

Florianópolis
01 de Dezembro de 2014

Agradecimentos

Aos meus pais, que me apoiaram incondicionalmente, compreendendo as eventuais ausências durante esses cinco anos, principalmente durante esta “reta final”.

Aos amigos, que me incentivaram e trouxeram alegria durante toda a graduação.

Aos colegas de graduação que sempre me ajudaram quando foi preciso.

A todos os professores e funcionários da UFSC, principalmente ao meu orientador, Rafael Holdorf Lopez, responsável pela transmissão do conhecimento indispensável para a realização desse trabalho.

E a todos que de alguma forma fizeram parte da minha formação, o meu muito obrigado.

Resumo

Na engenharia existem diversas etapas a serem seguidas até chegarmos ao produto final: pesquisa, análise, projeto, fabricação e venda. Muitas vezes, no processo de produção o engenheiro elabora um projeto que se adapte às especificações exigidas, de acordo com sua experiência, sem que se haja preocupação se a alternativa escolhida é realmente a melhor possível, ou seja, a que consome a menor quantidade de recursos.

A otimização estrutural visa a obtenção de uma solução ótima, transformando o projeto em um problema matemático, com o objetivo de se obter uma estrutura que atenda aos requisitos de desempenho impostos e, ao mesmo tempo, tenha o menor custo possível. Este método vem sendo muito aprimorado nas últimas décadas, devido ao avanço tecnológico dos computadores, que permite que a grande quantidade de processamento de dados e cálculos envolvidos no processo seja feita de maneira cada vez mais rápida e precisa.

Neste trabalho será feita a apresentação do conceito de otimização estrutural e do *Algoritmo Firefly (AF)*, um algoritmo metaheurístico que será utilizado para a otimização de projetos de treliças e comparado-o com outros métodos.

PALAVRAS-CHAVE: Otimização estrutural, Firefly Algorithm, algoritmo metaheurístico, treliças.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

No projeto de estruturas existem diversas variáveis a serem consideradas, como: lançamento da estrutura, determinando a posição dos elementos estruturais, escolha do material que será utilizado e dimensionamento de cada peça. Devido ao grande número de variáveis existentes, há o envolvimento de um grande número pessoas e um elevado consumo de tempo, que gera grandes custos. Para que haja economia de tempo, muitas vezes é adotado um método e se desenvolve o projeto a partir dele, sem se preocupar se a solução obtida é a melhor possível. Este método geralmente consiste em adotar um valor inicial (baseado na experiência do engenheiro) e, se este valor não atender aos requisitos a estrutura, modificá-lo até que se encontre um resultado o qual atenda todas as restrições impostas.

A evolução computacional e dos métodos numéricos das últimas décadas permitiu que os computadores realizassem cálculos complexos e processem uma grande quantidade de dados em um curto espaço de tempo. A engenharia pode se beneficiar desta evolução, adotando sistemas melhorados por meio da otimização das variáveis envolvidas no projeto.

A otimização consiste em minimizar ou maximizar o valor de uma função (chamada de função objetivo) e, ao mesmo tempo, atender as restrições impostas. Assim, uma estrutura pode ser formulada como um problema de otimização, sendo a função objetivo representada pelo seu custo, que desejamos que tenha o menor valor possível, e as restrições representadas pelos critérios estabelecidos pelas normas.

Segundo ARORA (2004, p. 3), o projeto deve ser um processo iterativo, consistindo em cinco etapas, ilustradas na Figura 1. Na *primeira etapa* devem ser definidas as especificações do sistema a ser desenvolvido, com interação entre o engenheiro e o proprietário, uma vez feito isso o projeto em si pode começar. A *segunda etapa* é a elaboração do projeto preliminar, estudando vários conceitos para o sistema em um curto período de tempo, por meio do uso de modelos

simplificados. Na *terceira etapa* deve ser escolhido um ou mais projetos preliminares, dentre os que apresentaram os melhores desempenhos e (usando um processo iterativo) detalhá-los, de maneira a maximizar o desempenho ou minimizar o custo. A *quarta etapa* é a fabricação de um protótipo do sistema e a *quinta etapa* é o teste deste protótipo, sendo esses dois últimos passos obrigatórios apenas quando o sistema for produzido em massa ou quando existem vidas humanas envolvidas. Após cada etapa do processo deve ser feita a avaliação do projeto, sendo necessário retornar a etapas anteriores caso alguma inconsistência seja encontrada.

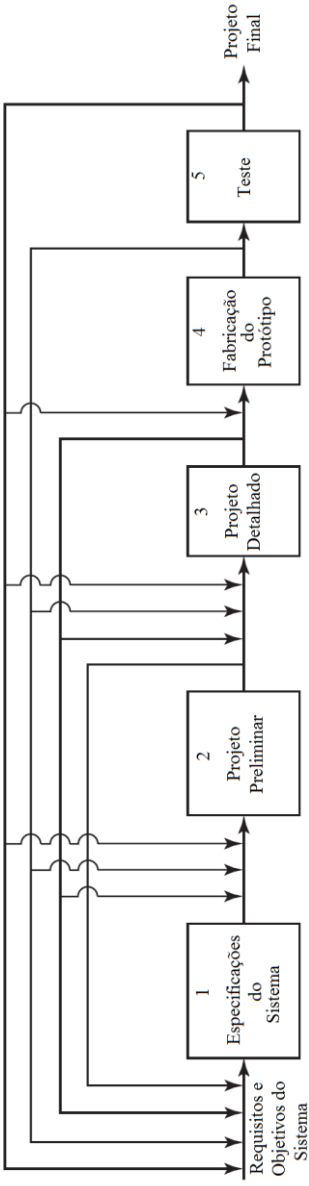


Figura 1 - Etapas de um Projeto

2. Objetivos

2.1. Objetivos Gerais

Este trabalho tem como objetivo a aplicação da otimização matemática de funções em problemas de engenharia, por meio do uso do *Algoritmo Firefly* (AF) e a comparação do seu desempenho com outros métodos de otimização.

2.2. Objetivos Específicos

Aplicação do *Algoritmo Firefly* (AF), identificando as variáveis de projeto, função objetivo e restrições do projeto para que seja encontrando o resultado que apresente o menor custo de execução e ainda assim atenda os requerimentos de desempenho.

Será dado enfoque em problemas de treliças, considerando a possibilidade de otimização dimensional (tamanho da seção de cada barra) e geométrica (posição dos nós), fazendo-se a comparação entre resultados obtidos com o algoritmo adotado e demais métodos de otimização, apresentados por diversos autores.

3. Otimização Estrutural

3.1. Formulação do Problema de Otimização

A otimização é um processo matemático que visa maximizar ou minimizar o valor de uma função, chamada de função objetivo. No caso de problemas de engenharia esta função geralmente é representada pelo custo, volume ou peso da estrutura.

Pode-se dividir a otimização estrutural em três tipos: Dimensional, onde se alteram as dimensões dos elementos; Geométrica, onde se altera a posição dos nós e Topológica, onde a estrutura em si é alterada, adicionando ou removendo elementos. Na Figura 2 está representada uma treliça e a seguir os três tipos de otimização que podem ser feitas:

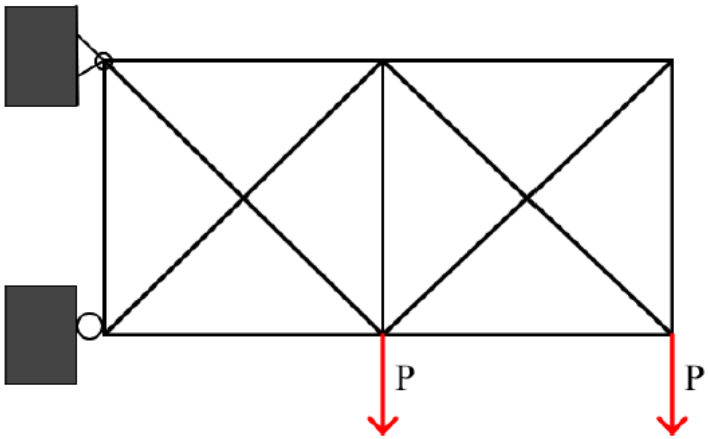


Figura 2 - Treliça Original (Fonte: BARBARESCO, 2014)

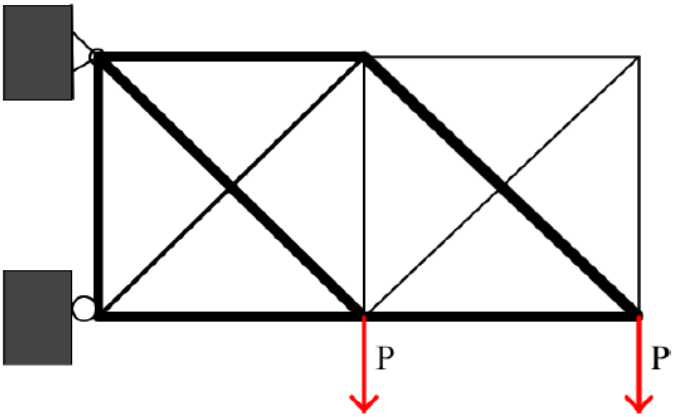


Figura 3 - Treliça Otimizada Dimensionalmente (Fonte: BARBARESCO, 2014)

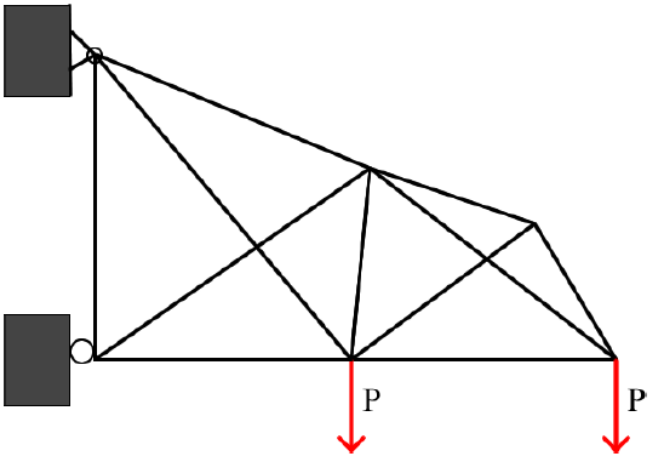


Figura 4 - Treliça Otimizada Geometricamente (Fonte: BARBARESCO, 2014)

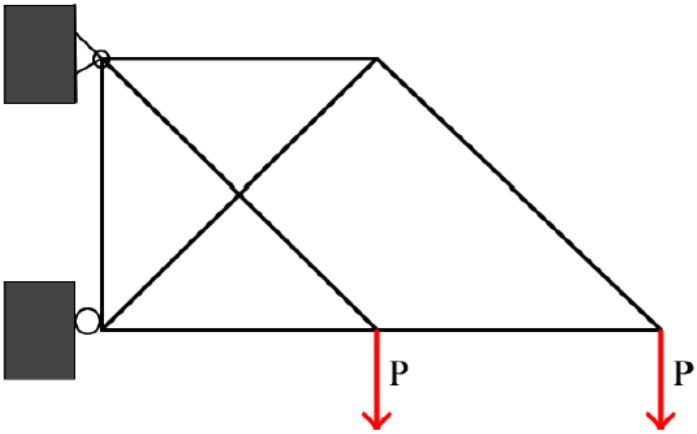


Figura 5 - Treliça Otimizada Topologicamente (Fonte: BARBARESCO, 2014)

A transformação de um problema de engenharia num problema matemático deve ser feita por meio dos seguintes passos: definição do projeto, coleta de dados, identificação das variáveis de projeto, identificação da função objetivo e identificação das restrições do projeto. A seguir descreve-se com detalhes esses cinco passos:

3.1.1. Definição do Projeto

A formulação do problema começa com a descrição do projeto, geralmente feita pelo proprietário da obra a ser construída, definindo quais os objetivos a serem alcançados e as restrições a serem cumpridas.

3.1.2. Coleta de Dados

Coleta de informações relevantes do projeto, como: requerimentos de desempenho, propriedades e custos dos materiais, esforços atuantes, dimensões máximas e mínimas. Estes dados serão as constantes do problema e caso não haja informação suficiente, devem ser feitas suposições para que ele possa ser formulado e resolvido.

Nesta fase também devem ser definidos os procedimentos e ferramentas a serem utilizados na análise e resolução do problema.

3.1.3. Identificação das Variáveis de Projeto

As variáveis de projeto são os parâmetros que identificam o sistema a ser montado. Elas são livres, podendo assumir qualquer valor dentro do domínio estipulado e devem ser independentes. Se houver dependência, deve-se ser criar equações que mostrem como elas se relacionam. O número de variáveis independentes determina o grau de liberdade do problema.

Se as variáveis de projeto não forem escolhidas de maneira adequada o resultado obtido pode ser incorreto, ou pode-se gerar um sistema sem solução. No início da formulação do problema é desejável que se crie mais variáveis de projeto que número de graus de liberdade existentes, para que haja mais flexibilidade na resolução do sistema. Posteriormente pode-se atribuir valor constante a algumas destas variáveis, transformando-as em dados do problema. Por exemplo: numa treliça metálica podem-se definir as seguintes variáveis de projeto: área da seção transversal das barras e posição dos nós.

3.1.4. Identificação da Função Objetivo

A função objetivo deve ser uma função escalar, determinada pelas variáveis de projeto e será o parâmetro a ser maximizado ou minimizado. Em problemas de engenharia as funções objetivos podem ser representadas por vários parâmetros: custo (que deve ser minimizado), peso (que deve ser minimizado), deslocamentos (que devem ser minimizados), resistência da peça (que deve ser maximizada), entre outros. Por exemplo: numa viga metálica, onde se deseja minimizar o custo, a função objetivo seria dada pelo produto do volume da peça (obtido através das dimensões da peça, que seriam as variáveis de projeto) com o preço por unidade volumétrica (que seria uma constante do problema).

3.1.5. Identificação das Restrições do Projeto

O último passo é identificar as restrições impostas pelo problema e definir uma expressão para cada uma delas, contendo pelo menos uma variável de projeto. Estas restrições podem ser impostas por critérios de projeto (como, por exemplo, a dimensão máxima de uma peça ou a flecha admissível) ou pelas características do material (como tensão máxima admissível) e devem ser descritas em função das variáveis de projeto.

3.2. Conceitos de Otimização

3.2.1. Admissibilidade do Projeto

Estando definida a função objetivo, qualquer vetor **x**, pertencente ao domínio da função é considerado solução do projeto, podendo esta ser admissível (se respeitar todas as restrições) ou não-admissível (se as restrições impostas não forem respeitadas).

Os valores que o vetor solução **x** deve assumir para que seja uma solução admissível, representado por **S**, é definido na equação **(1)**:

$$S = \{\mathbf{x} \mid \ h_i(\mathbf{x}) = 0, i = 1 \text{ a } p;\ \ g_j(\mathbf{x}) \leq 0, j = 1 \text{ a } m\} \qquad \textbf{(1)}$$

A equação **(1)** significa que os valores de **x** devem estar contidos no domínio **D** da função e respeitar as **p** restrições de igualdade (**h**) e as **m** restrições de desigualdade (**g**) existentes.

3.2.2. Solução Ótima

O vetor **x*** será a solução ótima se:

$$f(\mathbf{x}^*) < f(\mathbf{x}) \ \ \forall \ \ \mathbf{x} \in S \qquad \textbf{(2)}$$

Se a equação **(2)** for válida para todo domínio, **x*** é o ótimo global da função. Porém se ela for válida apenas para uma vizinhança **N** próxima a **x***, diz-se que **x*** é um mínimo local. **N** é definido da seguinte maneira:

$$N = \{\mathbf{x} \mid \mathbf{x} \in S; ||\mathbf{x} - \mathbf{x}^*|| < \delta\} \quad \textbf{(3)}$$

Onde **δ** é a distância a partir de **x*** analisada.

Considerando que desejemos minimizar o valor de f(**x**) na função representada pelo gráfico da Figura 6, com x podendo assumir valores entre **a** e **b**.

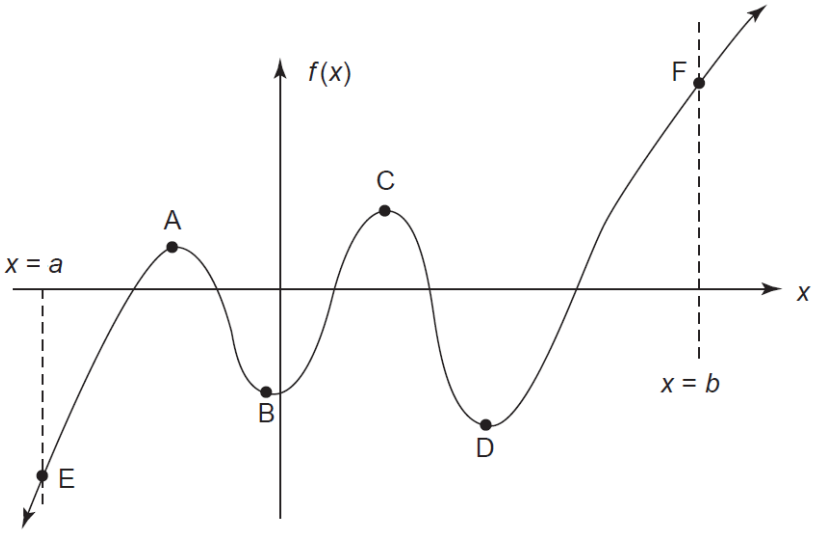


Figura 6 - Representação de uma função com ótimos locais e globais (Fonte: ARORA, 2004)

Podemos identificar os pontos B e D como ótimos locais, pois geram o menor valor de f(**x**) quando se observa apenas uma vizinhança próxima, e o ponto E como ótimo global, pois gera o menor valor de f(**x**) dentre todas as soluções adminssíveis.

3.3. Otimização Gráfica

Problemas de otimização com até duas variáveis podem ser resolvidos através da observação de seu gráfico juntamente com as restrições do problema, identificando região admissível e o ponto ótimo.

A seguir é apresentada a otimização gráfica do problema apresentado por ARORA (2004, p. 5), onde a função P deve ser maximizada:

Função Objetivo:

$P = 400*x_1 + 600*x_2$

Restrições:

g1: $x_1 + x_2 \leq 16$

g2: $x_1 + 2*x_2 \leq 28$

g3: $x_1/14 + x_2/24 \leq 1$

g4: $x_1 \geq 0$

g5: $x_2 \geq 0$

O primeiro passo é a representação das restrições, como mostrado na Figura 7, para que seja identificada a região admissível do problema:

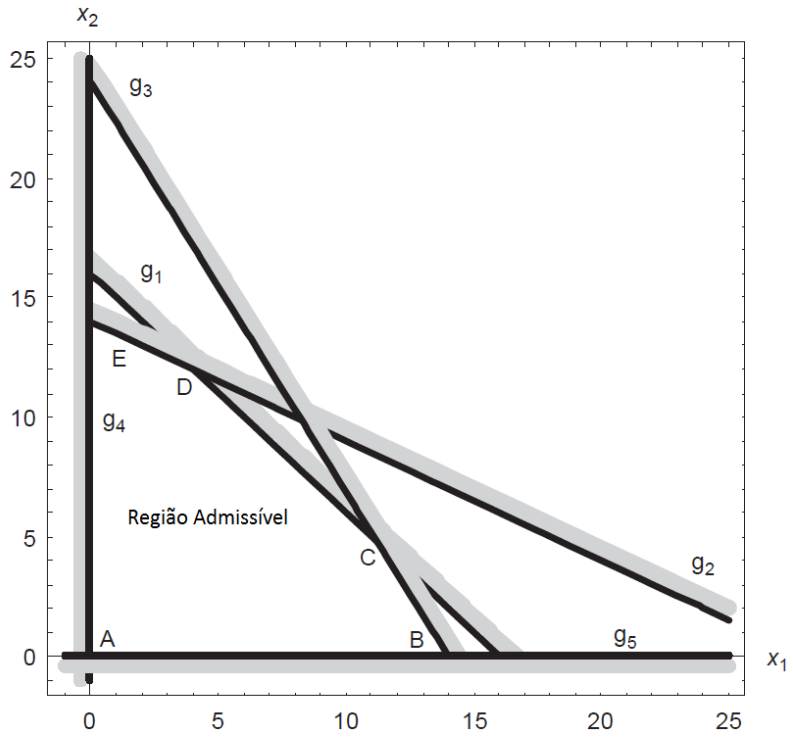


Figura 7 - Representação da Região Admissível, delimitada pelas 5 restrições do problema

A região admissível é o hexágono delimitado pelos pontos A, B, C, D e E.

Na sequência são plotadas as curvas de nível da função objetivo, arbritando valores dentro da região admissível para as variáveis $\mathbf{x_1}$ e $\mathbf{x_2}$.

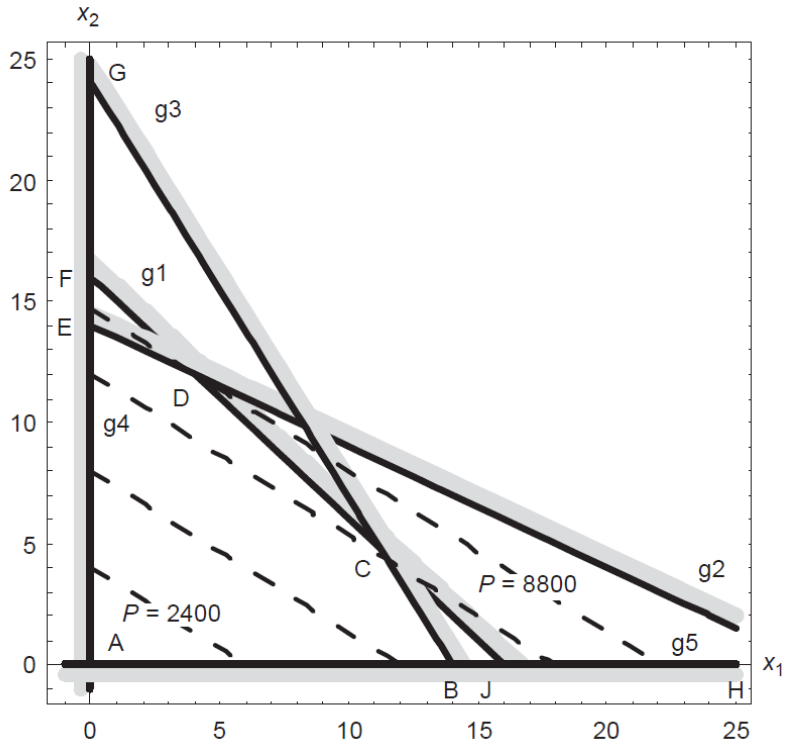


Figura 8 - Inclusão das Curvas de Nível da Função P

Observando a Figura 8 pode-se notar que o maior valor de **P** é 8800, pois essa curva de nível tangencia a região admissível no ponto **D**. Qualquer curva de nível de P com valor maior que 8800 estaria fora da região admissível. Portanto a Solução ótima é se dá no ponto D, onde:

$$\mathbf{P = 8800;\quad x_1 = 4;\quad x_2 = 12}$$

4. Classificação do Problema de Otimização

Um problema de otimização pode ser classificado como convexo (possuindo apenas um valor ótimo local, que também é o ótimo global) ou não-convexo (possui diversos ótimos locais), mas para classificar o problema em si precisamos dos conceitos de convexidade de conjuntos e funções.

Um conjunto pode ser classificado como convexo ou não-convexo, conforme ilustrado na Figura 9. O conjunto é considerado convexo quando: dados qualquer dois pontos pertencentes ao domínio da função, uma reta que ligue estes dois pontos deve estar inteiramente contida no domínio. Analogamente, uma função é considerada convexa se a união de quaisquer dois pontos pertencentes à função resulte numa linha que não ultrapasse o gráfico da função, como ser visto na Figura 10.

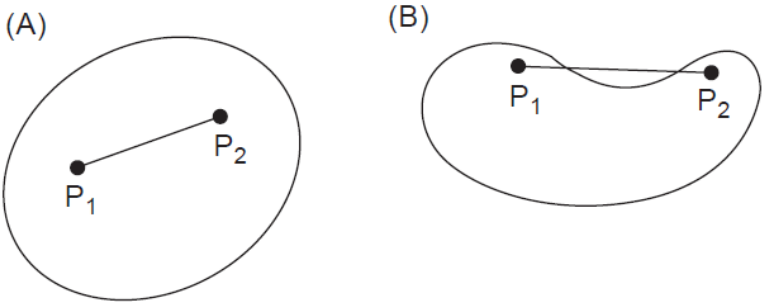


Figura 9 - Conjunto Convexo e Conjunto não-convexo

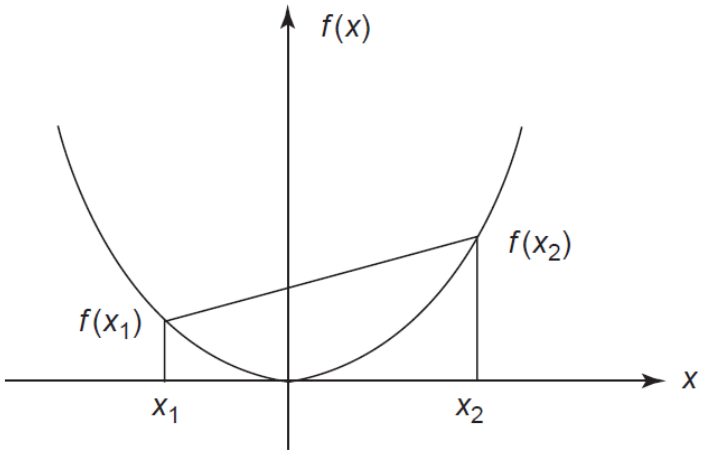


Figura 10 – Exemplo de Função Convexa

Esta verificação geométrica da convexidade de uma função fica impossibilitada quando estudamos funções mais complexas, com grande número de variáveis. Neste caso a convexidade pode ser checada da seguinte maneira: Sendo f(**x**) uma função qualquer, de **n** variáveis, onde **x** é um vetor com **n** componentes, f(**x**) é convexa apenas se a matriz Hessiana de f(**x**) for semidefinida ou definida positiva para todos os pontos do domínio admissível. A matriz Hessiana é definida pelas derivadas parciais de 2ª ordem da função.

Porém, em problemas de otimização devemos verificar a convexidade da região admissível e não apenas da função objetivo, pois uma função objetivo convexa pode gerar um problema não-convexo quando a analisamos juntamente com as restrições do projeto e vice-versa.

Finalmente, a análise da convexidade do problema de otimização é feita com a verificação conjunta da função objetivo e suas restrições, da seguinte forma: Seja **S** (equação **(1)** da seção 3.2.1) um conjunto definido pelas restrições de um problema de otimização, então **S** é

convexo apenas se as restrições de desigualdade (**g**) forem funções convexas e as restrições de igualdade (**h**) forem funções lineares.

Se o problema for convexo haverá apenas um valor mínimo possível, permitindo o uso de **métodos de otimização local**.

Já se o problema for não-convexo (como a maior parte dos problemas de engenharia) existirão vários valores mínimos locais, que faz com que tenhamos que utilizar **métodos de otimização global**.

4.1. Otimização Local

O algoritmo para busca do valor mínimo local consiste em se iniciar com o cálculo do valor da função em um ponto inicial ($\mathbf{x_o}$), contido no domínio possível e fazer iterações, modificando seu valor (em $\Delta\mathbf{x_i}$) e calculando o novo valor da função, com o intuito de otimizar o valor de $\mathbf{f(x)}$, até que o critério de parada seja atingido.

A variação Δx_i geralmente é calculada utilizando dois fatores: direção (representado por um vetor) e módulo (representado por um escalar) do deslocamento. A execução do algoritmo segue os seguintes passos:

1 – Estimar o valor inicial;

2 – Calcular uma direção para o deslocamento;

3 – Checar a convergência do algoritmo, se o valor mínimo foi alcançado o algoritmo para;

4 – Calcular o módulo do deslocamento;

5 – Calcular o novo valor da função, considerando a variação Δx_i;

6 – Retornar ao 2º passo.

A direção do deslocamento geralmente é calcula usando o vetor gradiente da função objetivo. Caso hajam restrições no problema, faz-se com que a direção do deslocamento seja perpendicular às restrições, quando x_i cair numa região não permitida, fazendo com que o valor volte a pertencer ao domínio possível.

4.2. Otimização Global

Os métodos de otimização global são divididos em dois grupos: determinísticos, que buscam o valor mínimo global da função objetivo através da consideração de critérios fixos, retornando sempre o mesmo resultado para uma determinada função; e estocásticos, que calculam o valor da função em diversos pontos e encontram o mínimo global com a combinação de certas regras e parâmetros aleatórios.

O método utilizado neste trabalho será um algoritmo estocástico e metaheurístico. Este tipo de algoritmo não é baseado em gradientes e é muito eficiente para a otimização de funções objetivo complexas. Partindo de um valor inicial, este método faz uma busca no domínio da função pelo valor mínimo global, seguindo determinados critérios e levando em conta informações de iterações anteriores, visando a convergência para o valor ótimo da maneira mais eficiente possível.

Os algoritmos metaheurísticos têm como base um fenômeno natural, onde a população inicial passa por modificações a cada iteração, analogamente à evolução dos organismos que ocorre na natureza. As vantagens deste método são: podem ser aplicados a funções descontínuas e não-lineares; não requerem que seja feita a análise do gradiente da função objetivo, obtendo ótimo desempenho mesmo em problemas onde o gradiente é difícil de ser calculado ou não exista; possuem vários parâmetros que podem ser calibrados para cada função, evitando que fique presos em ótimos locais. Porém, também apresenta algumas desvantagens: pode ser necessário um grande número de iterações para a otimização correta de funções mais complexas, resultando em um alto custo computacional; requerem ajuste de diversos parâmetros empíricos, que devem ser ajustados por tentativa e erro, variando de problema para problema.

5. Algoritmo Firefly

O algoritmo usado para a otimização de funções será o Algoritmo Firefly (AF). Trata-se de um algoritmo metaheurístico estocástico, baseado no comportamento dos vagalumes, desenvolvido por Xin-She Yang, da Universidade de Cambridge.

5.1. Comportamento dos Vagalumes

Na natureza os vagalumes usam seu brilho para atrair possíveis parceiros, sendo essa atração mais forte quanto maior for a intensidade da luz.

A ideia do algoritmo é calcular o valor da função objetivo em diversos pontos do domínio, escolhidos inicialmente de forma aleatória, considerando que em cada um destes pontos exista um vagalume e fazer com que o valor da função nestes pontos esteja relacionado com a intensidade da luz gerada pelos vagalumes. Em seguida são feitas iterações, seguindo certas regras, com o objetivo de fazer com que os valores convirjam para o ponto que gere o maior brilho, obtido no ponto onde a função apresente o valor ótimo (nos casos estudados este é o valor mínimo).

A atratividade está ligada à intensidade da luz **(I)** enxergada pelo vagalume, que diminui com o aumento da distância **(r)**, de maneira $I \propto r^{-2}$, portanto os pontos que apresentam os maiores valores para a função objetivo vão sendo atraídos em direção aos que apresentam os menores valores.

A partir das informações dadas acima são formuladas algumas regras para o algoritmo:

1) Um vagalume pode ser atraído por qualquer outro.
2) A atratividade é proporcional ao brilho. O vagalume de menor brilho sempre irá se mover em direção ao de maior brilho.

3) A atratividade é proporcional à intensidade da luz, que é inversamente proporcional à distância, portanto a atratividade diminuirá quanto maior for a distância entre os vagalumes.
4) Quando não houver nenhum outro com brilho maior que o seu, o vagalume se moverá de forma aleatória.
5) O brilho de um vagalume é determinado pela forma da função. Por exemplo: em um problema de minimização o brilho será inversamente proporcional ao valor da função, pois o algoritmo tem o objetivo de encontrar o ponto que a intensidade da luz seja máxima.

5.2. Atratividade

No FA existem dois pontos importantes: a intensidade da luz (ou brilho), que é um parâmetro individual de cada vagalume, e a atratividade, que depende da distância que o vagalume está sendo observado. Apesar de parecidos, são conceitos diferentes. Podemos assumir que a atratividade de um vagalume é proporcional ao seu brilho que, por sua vez, está associado à função a ser avaliada.

A intensidade da luz de um vagalume, pode ser escrita como **I(r)**, que diminui quanto mais longe estiver a fonte geradora, portanto a atratividade entre dois vagalumes também diminuirá se a distância entre eles aumentar. Considerando as reduções causadas pela distância e pela absorção do ar, temos a seguinte fórmula para a intensidade do brilho enxergada por um vagalume, em função da distância:

$$\mathbf{I(r) = I_0 e^{-\gamma r^2}} \qquad \textbf{(4)}$$

Onde $\mathbf{r}$ é a distância entre os vagalumes, $\boldsymbol{\gamma}$ é o coeficiente de absorção de luz, que determina quanto a intensidade diminui com a distância e $\mathbf{I_0}$ é a intensidade original do luz, em $\mathbf{r = 0}$.

Como a atratividade de um vagalume é proporcional à intensidade enxergada pelos vagalumes adjacentes, podemos definir a atratividade ($\boldsymbol{\beta}$) como:

$$\boldsymbol{\beta(r) = \beta_0 e^{-\gamma r^2}} \qquad \textbf{(5)}$$

Onde $\boldsymbol{\beta_0}$ é a atratividade em $\mathbf{r = 0}$.

Pela equação podemos definir a distância característica: $\boldsymbol{\Gamma = \gamma^{-1/2}}$,que é a distância necessária para a atratividade cair de $\boldsymbol{\beta_0}$ para $\boldsymbol{\beta_0 e^{-1}}$.

5.3. Movimento dos Vagalumes

A distância entre dois vagalumes (**i** e **j**) é dada por:

$$\mathbf{r_{ij} = ||x_i - x_j|| = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2}} \qquad \textbf{(6)}$$

Onde **n** é o número de dimensões nas quais o vagalume pode se deslocar.

A equação do movimento de um vagalume **i**, atraído por outro **j** é dado por:

$$\mathbf{x_i = x_{i0} + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_{i0}) + \alpha(rand - 1/2)} \qquad \textbf{(7)}$$

Onde:

$\mathbf{x_{i0}}$ é a posição inicial do vagalume **i**;

$\boldsymbol{\beta_0 e^{-\gamma r_{ij}^2}(x_j - x_{i0})}$ é a parcela do movimento devido à atratividade gerada pelo vagalume **j**;

$\boldsymbol{\alpha(rand - 1/2)}$ é a parcela aleatória do movimento, com **rand** sendo um número aleatório que pode variar entre 0 e 1.

n é o número de dimensões do vetor x.

O parâmetro $\boldsymbol{\gamma}$ é de extrema importância na determinação da velocidade de convergência do algoritmo. Teoricamente ele pode assumir qualquer valor positivo, mas na prática ele é determinado com base na distância característica (**Γ**) do sistema a ser otimizado. Seu valor geralmente varia entre 0,01 e 100.

Existem dois casos específicos importantes: quando $\boldsymbol{\gamma \to 0}$ e quando $\boldsymbol{\gamma \to \infty}$. Para $\boldsymbol{\gamma \to 0}$, a atratividade será sempre constante, o que

seria equivalente a vagalumes espalhados num céu ideal, onde todos podem ser observados, de qualquer distância e, portanto, sejam sempre atraídos em direção ao que apresenta a maior intensidade de luz. Esta situação faz com que apenas um valor ótimo seja atingido (o ótimo global) e corresponde a um caso especial do *Particle Swarm Optimization* (PSO), outro algoritmo de otimização. Com $\boldsymbol{\gamma} \rightarrow \infty$, temos uma situação completamente oposta: nenhum vagalume pode ser observado por outro, fazendo com que eles se movam de forma completamente aleatória. Este caso corresponde a um método de busca aleatória.

Como o parâmetro $\boldsymbol{\gamma}$ pode ser ajustado para que fique entre estes dois extremos, o Firefly apresenta um desempenho melhor que os dois métodos citados acima, podendo encontrar ótimos globais e locais de maneira muito efetiva

5.4. Validação

As funções usadas para o teste do desempenho dos algoritmos otimizadores são chamadas de Problemas Benchmark, que são funções multimodais intencionalmente complicadas, possuindo diversos pontos de máximos e mínimos locais, mas apenas um global.

Para testar seu desempenho, o AF será usado para encontrar o valor ótimo de Problemas Benchmark. Uma delas é a função de Michalewicz, definida pela equação **(8)**, com seu gráfico apresentado na Figura 11:

$$\mathbf{f(x)} = -\sum_{\mathbf{i=1}}^{\mathbf{d}} \mathbf{sen(x_i)} * [\mathbf{sen}(\frac{\mathbf{i} * \mathbf{x_i^2}}{\boldsymbol{\pi}})]^{\mathbf{2m}} \qquad \textbf{(8)}$$

Serão usados: **m = 10** e **d = 2**. Para estes parâmetros o valor ótimo é -1,8013, ocorrendo em (2,20; 1,57).

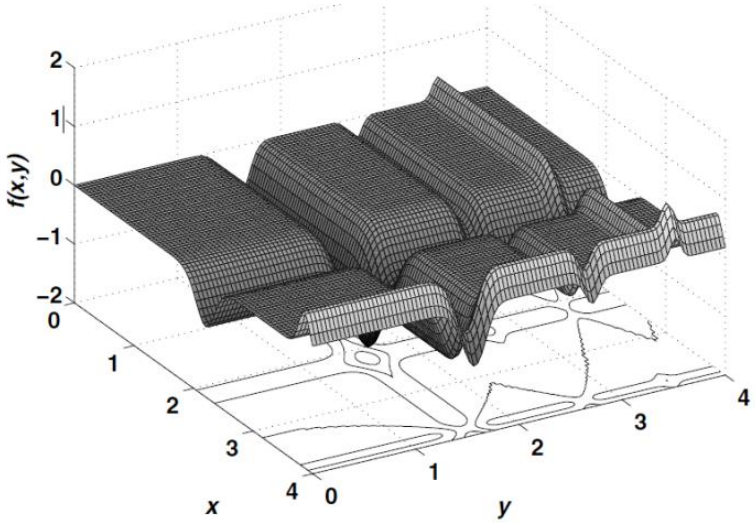


Figura 11 - Gráfico da Função de Michalewicz, para m=10 e d=2

O valor mínimo de $\mathbf{f(x)}$ encontrado foi -1,801,ocorrendo no ponto (2,2032; 1,5705).

Os parâmetros usados no algoritmo foram:

Número de Vagalumes = 40

Número de Iterações = 10

$\boldsymbol{\alpha}$ = 0,2

$\boldsymbol{\gamma}$ = 1

$\boldsymbol{\beta_0} = 1$

Na Figura 12 podem ser vistos os vagalumes nas posições iniciais e depois de 10 iterações:

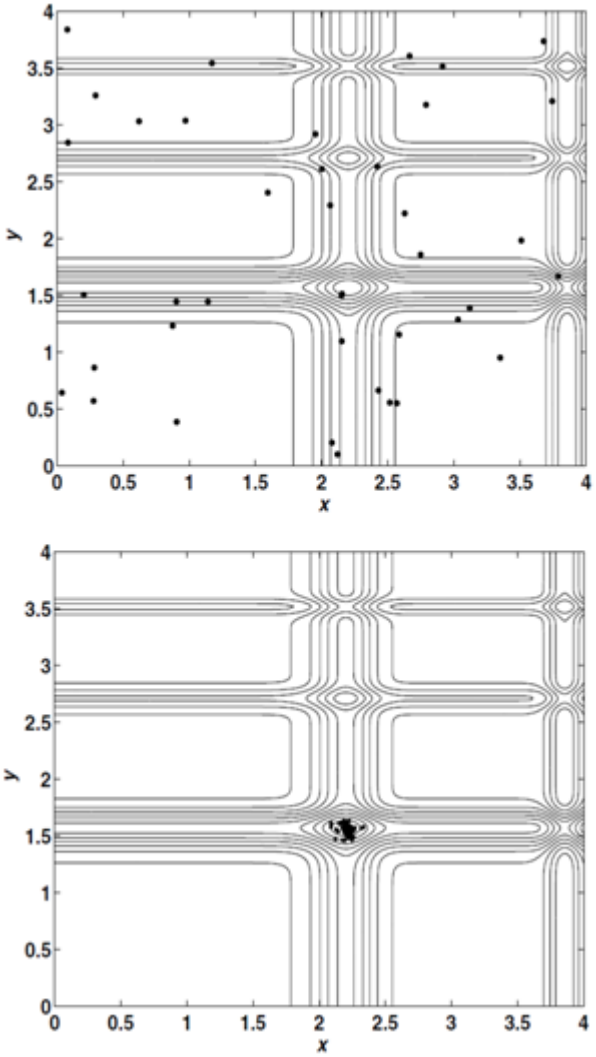


Figura 12 - Vagalumes nas suas posições iniciais (acima) e após 10 iterações (abaixo)

Nas Figuras 13 e 14 estão ilustradas outras duas funções utilizadas: a função de Yang e a função de Griewank, com apenas duas dimensões, para que possa ser feita a visualização de seus gráficos.

$$f(\mathbf{x}) = \left[e^{-\sum_{i=1}^{d}(x_i/a)^{2m}} - 2e^{-\sum_{i=1}^{d} x_i^2}\right] \cdot \prod_{i=1}^{d} \cos^2 x_i$$

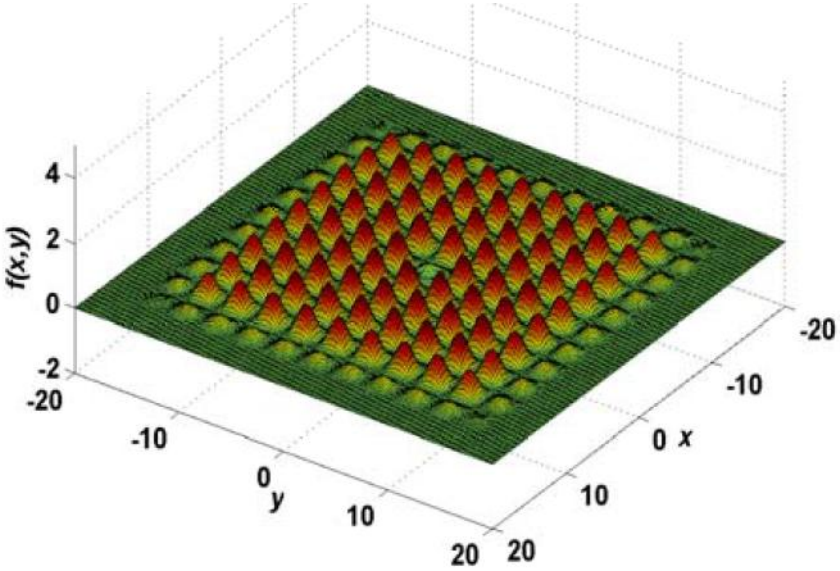


Figura 13 - Função de Yang, com d=2 e m=5.

$$f_n(\boldsymbol{x}) = \left(\frac{1}{4000}\right)\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

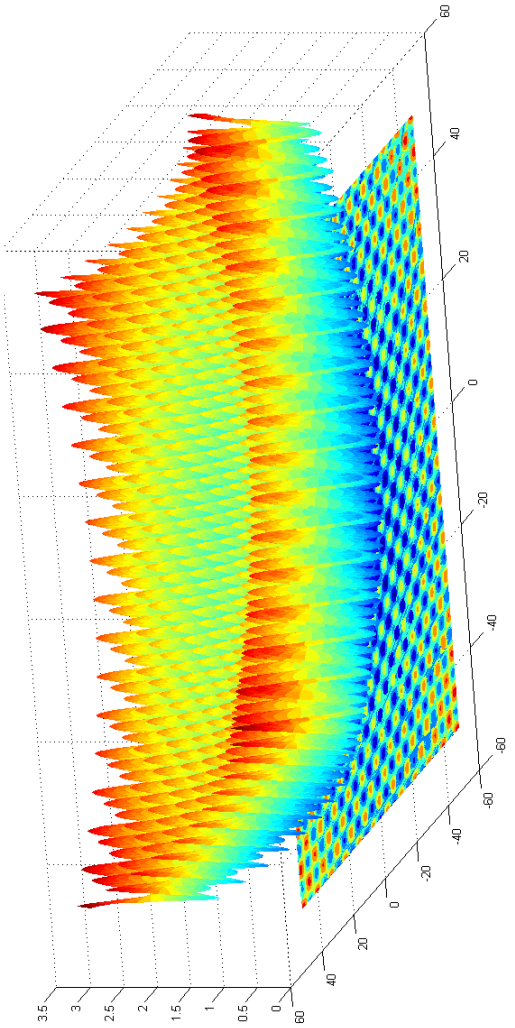


Figura 14 - Função de Griewank, com n=2.

A Tabela 1 faz a comparação dos resultados obtidos pelo Algoritmo Firefly com dois outros métodos: *Genetic Algorithm* (GA) e *Particle Swarm Optimization* (PSO), os valores contidos nela foram calculados por YANG (2010).

Tabela 1 - Comparação do desempenho AF com outros Métodos de Otimização, apresentada por YANG (2010).

Função	GA			PSO			AF		
	Nº de Avaliações	Desv. Padrão	Taxa de Acerto	Nº de Avaliações	Desv. Padrão	Taxa de Acerto	Nº de Avaliações	Desv. Padrão	Taxa de Acerto
Michalewicz (d=16)	89.325	7.914	95%	6.922	537	98%	3.752	725	99%
Rosenbrock (d=16)	55.723	8.901	90%	32.756	5.325	98%	7.792	2.923	99%
De Jong (d=256)	25.412	1.237	100%	17.040	1.123	100%	7.217	730	100%
Schwefel (d=128)	227.329	7.572	95%	14.522	1.275	97%	9.902	592	100%
Ackley (d=128)	32.720	3.327	90%	23.407	4.325	92%	5.293	4.920	100%
Rastrigin	110.523	5.199	77%	79.491	3.715	90%	15.573	4.399	100%
Easom	19.239	3.307	92%	17.273	2.929	90%	7.925	1.799	100%
Griewank	70.925	7.652	90%	55.970	4.223	92%	12.592	3.715	100%
Shubert	54.077	4.997	89%	23.992	3.755	92%	12.577	2.356	100%
Yang (d=16)	27.923	3.025	83%	14.116	2.949	90%	7.390	2.189	100%

Comparando os resultados podemos ver que o AF é muito mais eficiente em encontrar o ótimo global que os outros dois algoritmos apresentados, resultando em uma maior precisão e um menor número de avaliações.

5.5. Introdução de Restrições

Originalmente o AF não foi concebido para lidar com as restrições impostas pelo problema a ser otimizado, mas para isso podemos utilizar o Método da Penalização. Este método consiste em representar o valor final da função como define a equação **(9)**:

$$\mathbf{F(x) = f(x) + P} \qquad \textbf{(9)}$$

Onde P é a penalização aplicada, sendo nulo se nenhuma restrição for violada.

Seja $\mathbf{f(x)}$ uma função objetivo, com $\mathbf{h_i}$ restrições de igualdade e $\mathbf{g_j}$ restrições de desigualdade, será calculado um $\Delta\mathbf{g(x)}$ e um $\Delta\mathbf{h(x)}$, para cada valor de $\mathbf{x}$. Definidos pelas equações **(10)** e **(11)**:

$$\Delta\mathbf{g(x)} = \sum_{\mathbf{i=1}}^{\mathbf{m}} \mathbf{max}\{\mathbf{0}, \mathbf{g_j(x)}\} \qquad \textbf{(10)}$$

$$\Delta\mathbf{h(x)} = \sum_{\mathbf{i=1}}^{\mathbf{p}} \mathbf{max}\{\mathbf{0}, |\mathbf{h_i(x)}|\} \qquad \textbf{(11)}$$

As restrições de desigualdade devem ser do tipo $\mathbf{g_j} < 0$, para ser adicionada a penalização apenas se $\mathbf{g_j}$ tiver valor positivo. Já as restrições de Igualdade devem ser do tipo $\mathbf{h_i} = 0$, para ser adicionada a penalização apenas se $\mathbf{h_i}$ for diferente de zero (por isso a equação utiliza o módulo de $\mathbf{h_i}$, para que seja penalizado mesmo que o valor obtido seja negativo).

A penalização total é dada por:

$$\mathbf{P} = \boldsymbol{\lambda} * [\Delta\mathbf{g(x)} + \Delta\mathbf{h(x)}] \qquad \textbf{(12)}$$

Sendo $\boldsymbol{\lambda}$ um coeficiente que deve ser ajustado de forma empírica, para compensar as diferenças de ordem de grandeza entre os valores de $\mathbf{f(x)}$, $\mathbf{h_i}$ e $\mathbf{g_j}$.

Se todas as restrições forem respeitadas a penalização será nula.

6. Exemplos Numéricos

Em seguida serão apresentados dois exemplos de problemas resolvidos com o uso do AF, para a certificação de sua eficácia. O primeiro é uma função simples do segundo grau, onde o resultado obtido com o uso do AF será comparado com a solução analítica. O segundo é uma viga metálica, onde será feita a sua resolução por meio do AF e a comparação dos resultados com os obtidos por ARORA (2004).

6.1. Função de Segundo Grau

Minimizar a função do segundo grau com duas variáveis:

$$\mathbf{f(x) = x_1^2 + x_2^2 - 2.x_1 - 2.x_2}$$

Restrições:

$$\mathbf{x_1 + x_2 \leq 4}$$
$$\mathbf{x_1 \leq 2}$$
$$\mathbf{x_1 \geq 0}$$
$$\mathbf{x_2 \geq 0}$$

Parâmetros:

Número de Vagalumes = 20

Número de Iterações = 300

$\boldsymbol{\alpha}$ = 0,5

$\boldsymbol{\gamma}$ = 1

$\boldsymbol{\beta_0}$ = 0,5

$\boldsymbol{\lambda}$ = 100

Resultado obtido usando o AF:

u = [2,000 ; 1,008]

f(u) = - 0,999

Solução Analítica:

u = [2 ; 1]

f(u) = - 1

Nota-se que, por ser um método numérico, o algoritmo não retorna o valor exato, porém os resultados estão extremamente próximos ao valor real.

6.2. Viga Metálica

Determinar as dimensões ótimas de uma viga metálica de seção I, submetida à carga de um caminhão HS-20(MS18), especificado pela AASTHO (American Association of State Highway and Transportation Officials), dados:

Comprimeto da Viga (**L**) = 25 m

Módulo de Elasticidade do Material (**E**) = 210 GPa

Peso Específico do Material (**g**) = 77 kN/m³

Tensão de Escoamento ($\boldsymbol{\sigma}$) = 262 MPa

Tensão de Flexão Admissível ($\boldsymbol{\sigma_{adm}}$) = 144,1 MPa

Tensão de Cisalhamento Admissível ($\boldsymbol{\tau_{adm}}$) = 86,46 MPa

Tensão de Fadiga Admissível ($\boldsymbol{\sigma_{t,adm}}$) = 255 MPa

Flecha Admissível ($\mathbf{f_{adm}}$) = L/800

Carga Distribuída (**q**)= 19kN/m

Carga Concentrada, para o cálculo do Momento Fletor ($\mathbf{P_m}$) = 104kN

Concentrada, para o cálculo do Esforço Cortante ($\mathbf{P_s}$) = 155kN

Seção Transversal da viga:

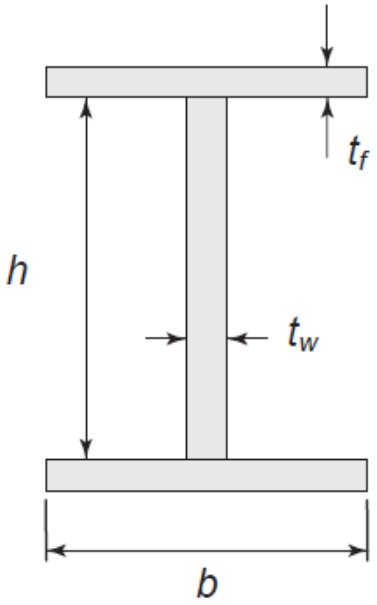


Figura 15 - Seção Transversal da Viga Metálica

Variáveis de Projeto:

h = Altura da Alma

$\mathbf{t_w}$ = Espessura da Alma

b = Largura da Mesa

$\mathbf{t_f}$ = Espessura da Mesa

Função Objetivo:

A função objetivo neste caso pode ser representada pelo volume total da viga, pois ele é diretamente proporcional ao seu custo:

$$\mathbf{Vol} = (\mathbf{h}.\mathbf{t_w} + \mathbf{b}.\mathbf{t_f}).\mathbf{L}$$

Variáveis Dependentes:

Área da Seção: $\mathbf{A} = \mathbf{h}.\mathbf{t_w} + \mathbf{2}.\mathbf{b}.\mathbf{t_f}$

Momento de Inércia: $\mathbf{I} = \frac{\mathbf{t_w}.\mathbf{h^3}}{\mathbf{12}} + \frac{\mathbf{2}.\mathbf{b}.\mathbf{t_f^3}}{\mathbf{3}} + \frac{\mathbf{b}.\mathbf{t_f}.\mathbf{h}.(\mathbf{h}+\mathbf{2}.\mathbf{t_f})}{\mathbf{2}}$

Carga Distribuída: $\mathbf{w} = (\mathbf{q} + \mathbf{g}.\mathbf{A})$

Momento Fletor: $\mathbf{M} = \frac{\mathbf{L}.(\mathbf{2}.\mathbf{P_m}+\mathbf{w}.\mathbf{L})}{\mathbf{8}}$

Tensão de Flexão: $\boldsymbol{\sigma} = \frac{\mathbf{M}.(\mathbf{0,5}.\mathbf{h}+\mathbf{t_f})}{\mathbf{1000}.\mathbf{I}}$

Flambagem da Mesa: $\boldsymbol{\sigma_m} = \mathbf{72\,845}.(\frac{\mathbf{t_f}}{\mathbf{b}})^2$

Flambagem da Alma: $\boldsymbol{\sigma_a} = \mathbf{3\,648\,276}.(\frac{\mathbf{t_w}}{\mathbf{h}})^2$

Esforço Cortante: $\mathbf{V} = \frac{(\mathbf{P_s}+\mathbf{w}.\mathbf{L})}{\mathbf{2}}$

Tensão de Cisalhamento: $\boldsymbol{\tau} = \frac{\mathbf{V}}{\mathbf{1000}.\mathbf{h}.\mathbf{t_w}}$

Flecha: $\mathbf{f} = \frac{\mathbf{L^3}.(\mathbf{8}.\mathbf{P_M}+\mathbf{5}.\mathbf{w}.\mathbf{L})}{\mathbf{384}.\mathbf{10^6}.\mathbf{E}.\mathbf{I}}$

Restrições:

Tensão de Flexão: $\boldsymbol{\sigma} \leq \boldsymbol{\sigma_{adm}}$

Flambagem da Mesa: $\boldsymbol{\sigma} \leq \boldsymbol{\sigma_m}$

Flambagem da Alma: $\boldsymbol{\sigma} \leq \boldsymbol{\sigma_a}$

Tensão de Fadiga: $\boldsymbol{\sigma \leq \frac{\sigma_{t,adm}}{2}}$

Tensão de Cisalhamento: $\boldsymbol{\tau \leq \tau_{adm}}$

Flecha: $\boldsymbol{f \leq f_{adm}}$

Limitações de Dimensão:

$0{,}30\text{m} \leq \boldsymbol{h} \leq 2{,}50\text{m}$

$0{,}01\text{m} \leq \boldsymbol{t_w} \leq 0{,}10\text{m}$

$0{,}30\text{m} \leq \boldsymbol{b} \leq 2{,}50\text{m}$

$0{,}01\text{m} \leq \boldsymbol{t_f} \leq 0{,}10\text{m}$

Resultado obtido usando o AF :

u = [2,0650 ; 0,0115 ; 0,3014 ; 0,0282] (m)

Vol(u) = 0,9051 m³

Estes Resultados foram muito parecidos (com uma diferença nas dimensões da mesa, mas com volumes totais muito próximos), que os obtidos por ARORA (2004, p. 373) para o mesmo exemplo, mesmo utilizando um algoritmo diferente para a resolução:

Resultado obtido por ARORA (2004):

u = [2,0753 ; 0,0115 ; 0,3960 ; 0,0156] (m)

Vol(u) = 0,9056 m³

7. Estruturas Treliçadas

Neste capítulo será feita a apresentação de alguns exemplos de otimização estrutural de treliças e a comparação dos resultados obtidos com o uso do AF com os apresentados por DEGERTEKIN et. al. (2013), MIGUEL et al. (2013), e LOPEZ et al. (2014).

7.1. Treliça com 10 barras – Otimização Dimensional

A seguir serão expostos dois exemplos resolvidos com o AF, fazendo apenas a otimização dimensional da treliça. Para ambos os casos temos a mesma estrutura, mudando apenas o valor das forças aplicadas (P_1 e P_2) nos nós 1, 2, 3 e 4. Os resultados foram comparados com os obtidos por diversos outros algoritmos, apresentados por DEGERTEKIN et. al. (2013). Os algoritmos usados foram:

PSO: *Particle Swarm Optimization*

PSOPC: *Particle Swarm Optimization + Passive Congregation*

HPSO: *Heuristic Particle Swarm Optimizer*

IHS: *Improved Harmony Search Algorithm*

ABC-AP: *Artificial Bee Colony Optimization*

EHS: *Efficient Harmony Search Algorithm*

SAHS: *Self-Adaptive Harmony Search Algorithm*

TBLO: *Teaching-Learning-Based Optimization*

A treliça a ser otimizada está representada na Figura 16, com as devidas dimensões e posicionamento das cargas. Os valores de seção transversal das barras podem variar, continuamente, entre 0,1pol² e 40 pol². A treliça está submetida a restrições de deslocamento e tensão, apresentados na Tabela 2, juntamente com outros parâmetros de projeto.

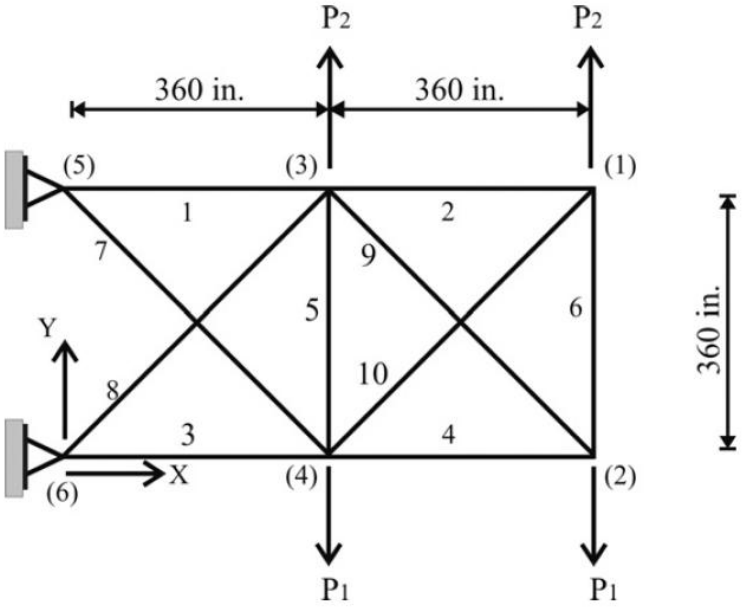


Figura 16 - Treliça do Exemplo 5.1, DEGERTEKIN et. al. (2013)

Tabela 2- Parâmetros do Exemplo 5.1, DEGERTEKIN et. al. (2013)

Parâmetro	**Valor**
Módulo de Elasticidade	10.000 ksi
Peso Específico	0,1 lb/pol^3
Tensão Admissível	25 ksi
Deslocamento Máximo	2 pol

Caso 1

Neste caso temos:

$P_1 = 100$ kips

$P_2 = 0$

A Tabela 3 mostra os resultados obtidos por diversos autores e apresentados por DEGERTEKIN et. al. (2013):

Tabela 3 - Resultados apresentados por DEGERTEKIN et. al. (2013) e obtidos com o AF para o Exemplo 5.1 – Caso 1

Variável de Projeto	Li et al.			Sonmez	Degertekin		Degertekin et al.	Este Trabalho
	PSO	PSOPC	HPSO	ABC-AP	EHS	SAHS	TBLO	AF
A_1 (pol²)	33,47	30,57	30,7	30,55	30,21	30,39	30,43	30,65
A_2 (pol²)	0,11	0,1	0,1	0,1	0,1	0,1	0,1	0,1
A_3 (pol²)	23,18	22,97	23,17	23,18	22,7	23,1	23,24	23,37
A_4 (pol²)	15,48	15,15	15,18	15,22	15,28	15,49	15,37	15,55
A_5 (pol²)	3,645	0,1	0,1	0,1	0,1	0,1	0,1	0,1
A_6 (pol²)	0,116	0,547	0,551	0,551	0,529	0,529	0,575	0,613
A_7 (pol²)	8,328	7,493	7,46	7,463	7,558	7,488	7,44	7,411
A_8 (pol²)	23,34	21,16	20,98	21,06	21,56	21,19	20,97	20,75
A_9 (pol²)	23,01	21,16	21,51	21,5	21,49	21,34	21,53	21,4
A_{10} (pol²)	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
Peso Ótimo	5529,5	5061	5060,9	5060,9	5062,4	5061,4	5061	5061,6
Peso Médio	-	-	-	-	5063,7	5062	5062,1	5063,8
Desv. Padrão	-	-	-	-	1,98	0,71	0,79	2,41

Foram usados os seguintes parâmetros para o AF:

Número de Vagalumes = 50

Número de Iterações = 3000

$\boldsymbol{\alpha}$ = 0,5

$\boldsymbol{\gamma}$ = 1

$\boldsymbol{\beta_0}$ = 0,1

$\boldsymbol{\lambda}$ = 10.000

Comparando os resultados pode-se notar que todos apresentam valores extremamente próximos, com exceção do PSO. O Algoritmo Firefly apresentou, comparativamente, bons resultados de peso ótimo e peso médio, gerando apenas um valor desvio padrão um pouco acima dos demais.

Caso 2

A única diferença deste caso em relação ao anterior são os módulos das forças, que agora são:

P_1 = 150 kips

P_2 = 50 kips

A Tabela 4 mostra os resultados apresentados por DEGERTEKIN et. al. (2013) e os obtidos usando o AF, com os mesmos parâmetros do Caso 1:

Tabela 4 - Resultados Apresentados por DEGERTEKIN et. al. (2013) e obtidos com o AF para o Exemplo 5.1 – Caso 2

Variável de Projeto	Li et al.			Sonmez	Degertekin		Degertekin et al.
	PSO	PSOPC	HPSO	ABC-AP	EHS	SAHS	TBLO
A_1 (pol²)	22,94	23,47	23,35	23,47	23,59	23,52	23,52
A_2 (pol²)	0,113	0,101	0,1	0,1	0,1	0,1	0,1
A_3 (pol²)	25,36	25,29	25,5	25,24	25,42	25,43	25,44
A_4 (pol²)	14,37	14,41	14,25	14,35	14,49	14,49	14,48
A_5 (pol²)	0,1	0,1	0,1	0,1	0,1	0,1	0,1
A_6 (pol²)	1,99	1,97	1,972	1,97	1,975	1,992	1,995
A_7 (pol²)	12,35	12,36	12,36	12,41	12,36	12,35	12,33
A_8 (pol²)	12,92	12,69	12,98	12,89	12,68	12,7	12,69
A_9 (pol²)	20,68	20,32	20,36	20,33	20,32	20,34	20,35
A_{10} (pol²)	0,1	0,103	0,101	0,1	0,1	0,1	0,1
Peso Ótimo	4679,5	4677,7	4677,3	4677,1	4679	4678,8	4681,2
Peso Médio	-	-	-	-	4681,6	4680,1	4680,1
Desv. Padrão	-	-	-	-	2,51	1,89	1,02

Assim como no Caso 1 os resultados apresentados foram bem próximos, desta vez incluindo o PSO, que era o único algoritmo que havia gerado um valor discrepante. Novamente Algoritmo Firefly apresentou bons

resultados de peso ótimo e peso médio e agora gerou o segundo menor valor de desvio padrão, ficando a atrás apenas do TBLO.

7.2. Treliça com 15 barras – Otimização Dimensional e Geométrica

A treliça de 15 barras, apresentada na Figura 17, será analisada duas vezes neste trabalho: uma considerando como restrição apenas a tensão de escoamento e outra considerado também a possibilidade de flambagem das barras comprimidas. Será feita a otimização dimensional e geométrica da treliça, comparando os resultados com os apresentados por MIGUEL et al. (2013), Problema 4.4.

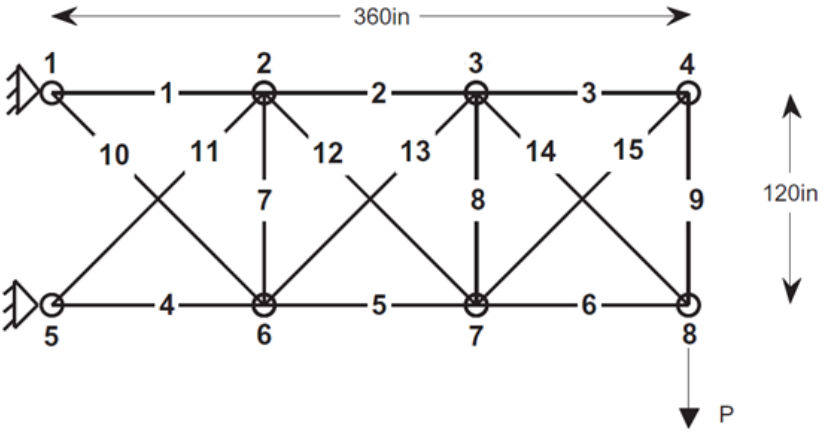


Figura 17 - Treliça do Exemplo 4.4, MIGUEL et. al. (2013)

A força P, aplicada no nó 8, tem módulo 10.000lb. Além das áreas das seções, as coordenadas horizontais e verticais dos nós 2, 3, 6 e 7 e coordenadas verticais dos nós 4 e 8 também são variáveis de projeto, no entanto as os nós 6 e 7 devem ter as mesmas coordenadas horizontais

que os nós 2 e 3, respectivamente. Portanto o problema apresenta 23 variáveis: 15 correspondentes às áreas das seções transversais de cada barra, 4 correspondentes às coordenadas verticais e horizontais dos nós 2 e 3, e 4 correspondentes às coordenadas verticais dos nó 4, 6, 7 e 8.

Este problema foi estudado em outras duas ocasiões por TANG et al. (2005) e WU et al. (1995), ambos utilizando o GA (*Genethic Algorithm*).

Os intervalos nos quais as coordenadas dos nós podem variar (em polegadas) são: $100 \leq x_2 \leq 140$; $220 \leq x_3 \leq 260$; $100 \leq y_2 \leq 140$; $100 \leq y_3 \leq 140$; $50 \leq y_4 \leq 90$; $-20 \leq y_6 \leq 20$; $-20 \leq y_7 \leq 20$; $20 \leq y_8 \leq 60$. Onde 'x_i' corresponde às coordenadas horizontais do nó e 'y_i' às verticais.

Além disso, as áreas das seções transversais das barras são variáveis discretas, podendo assumir os seguintes valores: (0,111; 0,141; 0,174; 0,220; 0,270; 0,287; 0,347; 0,440; 0,539; 0,954; 1,081; 1,174; 1,333; 1,488; 1,764; 2,142; 2,697; 2,800; 3,131; 3,565; 3,813; 4,805; 5,952; 6,572; 7,192; 8,525; 9,300; 10,850; 13,330; 14,290; 17,170; 19,180), em pol².

Trata-se de um problema de otimização de variáveis mistas, pois lida tanto com variáveis contínuas (coordenadas dos nós) quanto com variáveis discretas (seções das barras). Foram feitas alterações nas rotinas de cálculo para a adequação ao novo tipo de variável, mostrados no Anexo C (item 10.3).

Em nenhum dos casos serão verificados os deslocamentos nos nós da treliça.

As propriedades do material estão na Tabela 5:

Tabela 5 - Propriedades do Material, Exemplo 4.4 MIGUEL et. al. (2013)

Propriedade	**Valor**
Módulo de Elasticidade	10.000 ksi
Peso Específico	0,1 lb/pol^3
Tensão Admissível	25 ksi

Caso 1: Otimização Dimensional e Geométrica, desconsiderando a flambagem.

Neste caso a única restrição imposta será a tensão nas barras, que deve ser menor que a tensão admissível.

Na Tabela 6 é feita a comparação entre os resultados apresentados por MIGUEL et. al. (2013) e os obtidos neste trabalho, utilizando 8000 análises (com 10 vagalumes e 800 iterações), a mesma quantidade dos trabalhos apresentados, para que seja feita uma comparação coerente.

Tabela 6 - Resultados Apresentados por MIGUEL et. al. (2013) e obtidos com o AF para o Exemplo 4.4 – Caso 1

Variável de Projeto	Tang et al.			Wu et al.	Miguel et al.	Este Trabalho
	GA 1	GA 2	GA 3	GA	AF	AF
A_1 (pol²)	1,081	0,954	1,081	1,174	0,954	0,954
A_2 (pol²)	0,954	0,954	0,539	0,954	0,539	0,539
A_3 (pol²)	0,111	0,111	0,287	0,44	0,22	0,27
A_4 (pol²)	1,174	1,174	0,954	1,333	0,954	0,954
A_5 (pol²)	0,539	2,697	0,954	0,954	0,539	0,539
A_6 (pol²)	0,539	0,539	0,22	0,174	0,22	0,22
A_7 (pol²)	0,954	0,111	0,111	0,44	0,111	0,247
A_8 (pol²)	0,22	0,111	0,111	0,44	0,111	0,111
A_9 (pol²)	0,539	0,111	0,287	1,081	0,287	0,247
A_{10} (pol²)	0,287	0,539	0,22	1,333	0,44	0,44
A_{11} (pol²)	0,539	0,111	0,44	0,174	0,44	0,247
A_{12} (pol²)	0,111	0,111	0,44	0,174	0,22	0,287
A_{13} (pol²)	0,287	0,539	0,111	0,347	0,22	0,22
A_{14} (pol²)	0,539	0,539	0,22	0,347	0,27	0,22
A_{15} (pol²)	0,27	0,111	0,347	0,44	0,22	0,27
x_2 (pol)	134,94	139,67	133,61	123,19	114,97	130,77
x_3 (pol)	252,44	220,68	234,75	231,6	247,04	247,02
y_2 (pol)	125,89	115,73	100,45	107,19	125,92	125,95
y_3 (pol)	100,37	106,47	104,74	119,18	111,07	116,84
y_4 (pol)	80,72	53,01	73,76	60,46	58,3	63,04
y_6 (pol)	10,39	16,36	10,07	-16,73	-17,56	-3,27
y_7 (pol)	0,5	12,26	-1,34	15,56	-5,82	6,61
y_8 (pol)	28,52	43,69	50,4	36,64	31,46	57,84
Peso Ótimo	106,01	100,33	79,82	120,53	75,55	75,59
Peso Médio	-	-	-	-	84,45	83,75
Coef. de Variação	-	-	-	-	6,49	4,84

Fazendo a comparação nota-se que o AF apresenta desempenho consideravelmente melhor que os quatro tipos de GA apresentados neste problema. Os resultados obtidos apresentaram uma pequena diferença em relação aos de MIGUEL et. al. (2013), mesmo sendo utilizado o mesmo algoritmo e número de avaliações. Isto se deve a dois motivos: a natureza estocástica do algoritmo, que faz com que o resultado dependa de variáveis aleatórias e os parâmetros (α, β_0, γ e λ) adotados para a resolução podem ter sido diferentes.

Caso 2: Otimização Dimensional e Geométrica, considerando a flambagem.

Neste caso além da restrição de tensão nas barras, considerada anteriormente, ainda será verificada a estabilidade das barras submetidas a compressão. Para que não haja flambagem da barra o valor da força atuante deve estar abaixo da tensão crítica de Euler (σ_{cr}), apresentada na equação **(13)**:

$$\sigma_{cr} = \frac{100 \cdot E \cdot A_i}{8 \cdot L_i^2} \qquad \textbf{(13)}$$

As propriedades do material, os valores admissíveis de seção transversal das barras e o intervalo de variação das coordenadas dos nós são os mesmos do Caso 1.

A Tabela 7 mostra a comparação dos novos resultados, comparando apenas com os obtidos por WU et al. (1995) e MIGUEL et al. (2013), pois TANG et al. (2005) não apresentou dados com consideração de flambagem.

Tabela 7 - Resultados Apresentados por MIGUEL et. al. (2013) e obtidos com o AF para o Exemplo 4.4 – Caso 2

Variável de Projeto	Wu et al.	Miguel et al.	Este Trabalho
	GA	AF	AF
A_1 (pol²)	5,952	1,174	1,081
A_2 (pol²)	1,764	1,081	0,954
A_3 (pol²)	1,488	0,44	0,287
A_4 (pol²)	5,952	1,764	1,764
A_5 (pol²)	2,142	1,488	1,333
A_6 (pol²)	1,488	1,081	0,954
A_7 (pol²)	2,8	0,111	0,111
A_8 (pol²)	1,333	0,22	0,247
A_9 (pol²)	0,347	1,488	0,539
A_{10} (pol²)	1,081	0,174	0,287
A_{11} (pol²)	1,333	0,22	0,247
A_{12} (pol²)	1,081	0,22	0,247
A_{13} (pol²)	1,333	1,333	1,764
A_{14} (pol²)	1,764	0,287	0,287
A_{15} (pol²)	1,764	1,333	1,081
x_2 (pol)	118,89	109,56	112,96
x_3 (pol)	222,57	224,87	227,58
y_2 (pol)	108,02	103,31	106,67
y_3 (pol)	124,07	100,44	119,65
y_4 (pol)	54,84	51,58	58,39
y_6 (pol)	0,35	17,06	14,23
y_7 (pol)	15,52	19,02	10,62
y_8 (pol)	34,71	48,85	34,72
Peso Ótimo	402,36	138,07	139,9
Peso Médio	-	154,21	151,04
Coef. de Variação	-	3,85	3,11

Os resultados obtidos por este trabalho foram novamente parecidos com os obtidos por MIGUEL et al. (2013), pelos mesmos motivos do caso anterior.

Podemos perceber ainda um enorme diferença entre os resultados gerados pelo AF e os encontrados por WU et al. (1995). Isso se deve aos grandes avanços que ocorreram nos dezoito anos de diferença entre os trabalhos, tanto nos recursos tecnológicos, que permitem que seja feita uma quantidade maior de cálculos em um intervalo de tempo menor, quanto nos próprios algoritmos, que vêm sendo aprimorados e com isso convergindo de forma cada vez mais eficiente ao valor ótimo.

7.3. Treliça com 21 barras com 3 casos de carregamento – Otimização Dimensional e Geométrica

Para este último exemplo será estudada uma treliça de 21 barras, apresentada na Figura 18, com três casos de carregamento e considerando como restrições: as tensões nas barras, os deslocamentos nos nós e a flambagem das barras submetidas à compressão. Será feita a otimização dimensional e geométrica da treliça, comparando os resultados com os apresentados por LOPEZ et al. (2014), Problema 5.2.

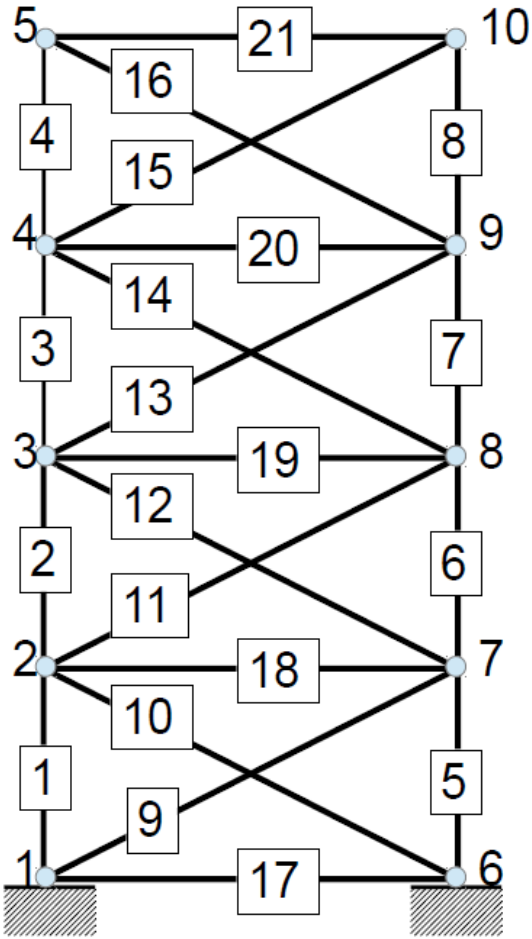


Figura 18 - Treliça do Exemplo 5.2, LOPEZ et. al. (2014)

Os três casos de carregamento a serem analisados estão ilustrados na Figura 19:

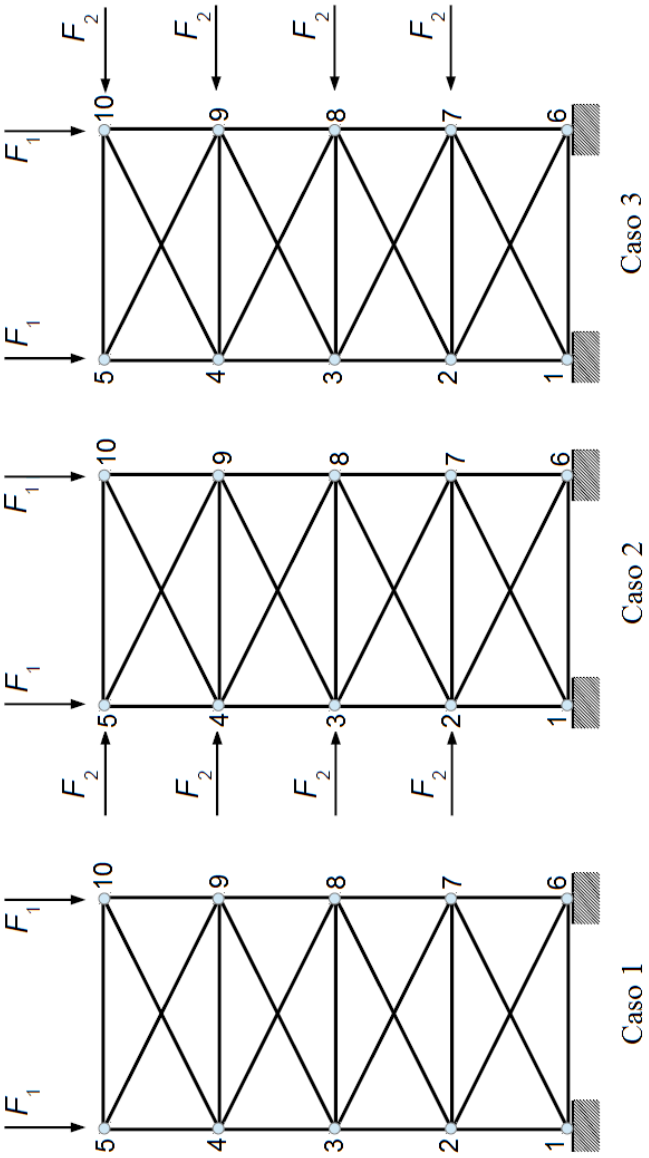


Figura 19 - Casos de Carregamento do Exemplo 5.2, LOPEZ et al. (2014)

Os carregamentos F_1 e F_2 tem módulos de 100kN e 25kN, respectivamente.

A treliça tem altura de 40m, com as posições verticais dos nós fixas e uniformemente espaçados em 10m. A base mede 20m, sendo apenas a posição dos nós 1 e 6 fixas, os demais nós podem se mover horizontalmente, mas sempre mantendo a simetria em relação ao eixo vertical. Devido a esta simetria e à simetria dos carregamentos podemos notar que as áreas das barras verticais e diagonais serão iguais, aos pares (A_1=A_5, A_2=A_6, A_3=A_7, A_4=A_8, A_9=A_{10}, A_{11}=A_{12}, A_{13}=A_{14}, A_{15}=A_{16},).

As variáveis de projeto são as áreas das barras 1, 2, 3, 4, 9, 11, 13, 15, 17, 18, 19, 20 e 21 e as coordenadas horizontais dos nós 2, 3, 4 e 5 (as coordenadas dos nós 7, 8, 9 e 10 também variam de acordo com estas últimas quatro, para que a simetria seja mantida). Portanto o problema apresenta 17 variáveis: 13 correspondentes às áreas das seções transversais de cada barra, 4 correspondentes às coordenadas dos nós.

As coordenadas horizontais dos nós podem variar até 8m em relação ao projeto inicial. As áreas disponíveis para as barras são: (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370 e 380), em cm².

Assim como o exemplo anterior trata-se de um problema de otimização de variáveis mistas, por lidar com variáveis contínuas e variáveis discretas. A rotina utilizada para o cálculo foi a contida no Anexo C (item 10.3), adaptando-se apenas os parâmetros lá especificados para a adequação ao novo problema.

O momento de inércia (**I**) de cada barra, usado na verificação da estabilidade, se relaciona com área da seguinte forma:

$$\mathbf{I = 0,36 * A^2}$$

O deslocamento máximo que cada nó pode sofrer é 1cm, tanto na horizontal, quanto na vertical.

A treliça é feita de aço, com as seguintes propriedades:

- Densidade de 8002kg/m³, que equivale a 78,42kN/m³.
- Módulo de Elasticidade de 210 GPa.
- Tensão de Escoamento de 210 MPa.

Para obtenção dos parâmetros de projeto foram usados coeficientes para majoração das cargas minoração do módulo de elasticidade, da tensão admissível e das áreas das barras, resultando nos valores apresentados na Tabela 8:

Tabela 8 - Parâmetros de Projeto para o Exemplo 5.2 LOPEZ et al. (2014)

Parâmetro	Valor Característico	Coeficiente	Valor de Projeto
F_1	100 kN	1,1	130 kN
F_2	25 kN	1,3	32,5 kN
Módulo de Elasticidade	210 GPa	0,9	189 GPa
Tensão Admissível	210 MPa	0,9	189 MPa
Área da Seção	-	0,95	0,95*A

Na Tabela 9 é feita a comparação entre os resultados apresentados por LOPEZ et. al. (2014), que fez a comparação entre três algoritmos diferentes: GA (Genetic Algorithm), HS (Harmonic Search) e o AF. Os valores obtidos neste trabalho foram alcançados utilizando 60.000 análises (com 20 vagalumes e 3.000 iterações), a mesma quantidade dos trabalhos apresentados.

Tabela 9 - Resultados Apresentados por LOPEZ et. al. (2014) e obtidos com o AF para o Exemplo 5.2

Variável de Projeto	Lopez et al.			Este Trabalho
	GA	HS	AF	AF
A_1, A_5 (cm^2)	110	100	100	90
A_2, A_6 (cm^2)	90	110	90	80
A_3, A_7 (cm^2)	70	60	70	70
A_4, A_8 (cm^2)	40	40	40	50
A_9, A_{10} (cm^2)	60	60	60	60
A_{11}, A_{12} (cm^2)	40	40	40	50
A_{13}, A_{14} (cm^2)	50	50	50	40
A_{15}, A_{16} (cm^2)	10	10	10	10
A_{17} (cm^2)	10	10	10	10
A_{18} (cm^2)	10	10	10	10
A_{19} (cm^2)	10	10	10	10
A_{20} (cm^2)	10	10	10	10
A_{21} (cm^2)	20	10	20	10
x_2 (cm)	226,1	251,1	228	238,3
x_3 (cm)	429,3	417,9	434	430,6
x_4 (cm)	578,7	589	557,4	599,5
x5 (cm)	791,8	751,5	800	800
Peso Ótimo (kN)	105,1	104,6	103,6	103,7
Peso Médio (kN)	114,2	113,9	110,4	110,8
Desv. Padrão (kN)	8,5	7,2	5,6	4,6

Analisando os resultados percebe-se que o AF gerou valores menores que os outros dois algoritmos para todos os parâmetros, isso tanto para a solução obtida por LOPEZ et al. (2013) quanto pela obtida por este trabalho, comprovando mais ou vez sua eficácia.

8. Conclusões

Neste trabalho foram apresentados conceitos sobre otimização estrutural, como: tipos de otimização possíveis, processo para transformação de um projeto de engenharia em um problema de otimização e suas classificações. Levando em conta estes conceitos optou-se pelo uso do AF (Algoritmo Firefly) para a resolução dos problemas de otimização propostos, pelo fato dele ser um algoritmo estocástico, que pode ser utilizado para a otimização global de problemas não-convexos (que é o caso da maioria dos problemas de engenharia), diferentemente dos métodos de otimização local baseados em gradiente.

O desempenho do algoritmo foi muito satisfatório comparado a outros métodos de otimização em alguns dos casos analisados, retornando sempre resultados muito próximos aos ótimos encontrados na literatura consultada, como no exemplo 7.1 e até menores, como nos exemplos 6.2, 7.2 e 7.3.

Pode-se notar ainda a grande evolução ocorrida nesta área nas últimas duas décadas observando-se os resultados do exemplo 7.2: para o mesmo problema foi obtida uma estrutura com menos da metade do peso da obtida usando o GA (*Genetic Algorithm*), num estudo de dezenove anos atrás.

Durante a resolução dos problemas também se percebeu a importância dos parâmetros do algoritmo, que apesar de serem determinados de forma empírica podem diminuir o desempenho e até gerar soluções fora da região admissível do problema se usados valores incompatíveis com a ordem de grandeza do problema.

9. Referências

YANG, Xin-She. **Firefly Algorithm for Multimodal Optimization.** University of Cambridge, Cambridge (2010).

ARORA, Jasbir S. **Introduction to Optimum Design.** Elsevier, 2nd Edition (2004).

DEGERTEKIN, S. O.; HAYALIOGLU, M. S. **Sizing truss structures using teaching-learning-based optimization**. Computers and Structures 119 (2013) 177-188.

MIGUEL, Leandro F. F.; LOPEZ, Rafael H.; MIGUEL, Letícia F. F. **Multimodal size, shape, and topology optimisation of truss structures using the Firefly algorithm**. Advances in Engineering Software 56 (2013) 23-37.

LOPEZ, Rafael H.; TORII, André J.; MIGUEL, Leandro F. F.; CURSI, José E. S. de. **An approach for the global reliability based optimization of the size and shape of truss structures.** Federal University of Santa Catarina, Florianópolis (2014).

CRUZ, André G. F. **Otimização de Estruturas: Método para Otimização Global de Problemas**. Universidade Federal de Santa Catarina, Florianópolis (2012).

BARBARESCO, Guilherme M. **Otimização de Problemas de Engenharia Utilizando o Algoritmo da Competição Imperialista (ICA).** Universidade Federal de Santa Catarina, Florianópolis (2014).

ASSIS, Alysson R. M. G. de. **Adaptação do algoritmo Backtracking Search Optimization para o tratamento de problemas com restrições**. Universidade Federal de Santa Catarina, Florianópolis (2014).

McGUIRE, William; GALLAGHER, Richard H.; ZIEMIAN, Ronald D. **Matrix Structural Analysis**. John Wiley & Sons, 2nd Edition (2000).

DEGERTEKIN, S. O. **An improved harmony search algorithms for sizing optimization of truss structures**. Comput and Struct (2012); 92-93:229-41.

LI, L. J.; HUANG, Z. B.; LIU, F.; WU Q. H. **A heuristic particle swarm optimizer for optimization of pin connected structures.** Comput and Struct (2007); 85:340-9.

SONMEZ, M. **An Artificial bee colony algorithm for optimization of truss optimization**. Appl Soft Comput (2011); 11:2406-18.

WU S. J.; CHOW P.T. **Integrated discrete and configuration optimization of trusses using genetic algorithms**. Comput and Struct (1995); 56(4):695-702.

TANG W.; TONG, L.; GU Y. **Improved genetic algorithm for design optimization of truss structures with sizing, shape and topology variables**. Int J Numer Meth Eng (2005); 62:1737-62.

10. Anexos

10.1. ANEXO A – Algoritmo Firefly original, em MATLAB

```
%
=================================================
===== %
% Files of the Matlab programs included in the
book:       %
% Xin-She Yang, Nature-Inspired Metaheuristic
Algorithms,  %
% Second Edition, Luniver Press, (2010).
www.luniver.com %
%
=================================================
===== %

% -----------------------------------------------
------- %
% Firefly Algorithm for constrained optimization
using     %
% for the design of a spring (benchmark)
%
% by Xin-She Yang (Cambridge University) Copyright
@2009   %
% -----------------------------------------------
------- %

function fa_mincon
% parameters [n N_iteration alpha betamin gamma]
para=[40 500 0.5 0.2 1];
format long;

help fa_mincon.m
% This demo uses the Firefly Algorithm to solve the
% [Spring Design Problem as described by Cagnina et
al.,
% Informatica, vol. 32, 319-326 (2008). ]
```

```
% Simple bounds/limits
disp('Solve the simple spring design problem ...');
Lb=[0.05 0.25 2.0];
Ub=[2.0 1.3 15.0];


% Initial random guess
u0=Lb+(Ub-Lb).*rand(size(Lb));

[u,fval,NumEval]=ffa_mincon(@cost,@constraint,u0,Lb
,Ub,para);

% Display results
bestsolution=u
bestojb=fval
total_number_of_function_evaluations=NumEval

%%% Put your own cost/objective function here -----
---%%%
%% Cost or Objective function
 function z=cost(x)
z=(2+x(3))*x(1)^2*x(2);

% Constrained optimization using penalty methods
% by changing f to F=f+ \sum lam_j*g^2_j*H_j(g_j)
% where H(g)=0 if g<=0 (true), =1 if g is false

%%% Put your own constraints here ----------------
---%%%
function [g,geq]=constraint(x)
% All nonlinear inequality constraints should be
here
% If no inequality constraint at all, simple use
g=[];
g(1)=1-x(2)^3*x(3)/(71785*x(1)^4);
% There was a typo in Cagnina et al.'s paper,
% the factor should 71785 instead of 7178!
tmpf=(4*x(2)^2-x(1)*x(2))/(12566*(x(2)*x(1)^3-
x(1)^4));
g(2)=tmpf+1/(5108*x(1)^2)-1;
g(3)=1-140.45*x(1)/(x(2)^2*x(3));
```

```
g(4)=x(1)+x(2)-1.5;

% all nonlinear equality constraints should be here
% If no equality constraint at all, put geq=[] as
follows
geq=[];

%%% End of the part to be modified ----------------
---%%%

%%% ------------------------------------------------
---%%%
%%% Do not modify the following codes unless you
want %%%
%%% to improve its performance etc
%%%
% ------------------------------------------------
------
% ===Start of the Firefly Algorithm Implementation
======
% Inputs: fhandle => @cost (your own cost function,
%                    can be an external file  )
%     nonhandle => @constraint, all nonlinear
constraints
%                    can be an external file or a
function
%         Lb = lower bounds/limits
%         Ub = upper bounds/limits
%   para == optional (to control the Firefly
algorithm)
% Outputs: nbest   = the best solution found so far
%          fbest   = the best objective value
%      NumEval = number of evaluations:
n*MaxGeneration
% Optional:
% The alpha can be reduced (as to reduce the
randomness)
% ------------------------------------------------
--------

% Start FA
function [nbest,fbest,NumEval]...
```

```
           =ffa_mincon(fhandle,nonhandle,u0, Lb,
Ub, para)
% Check input parameters (otherwise set as default
values)
if nargin<6, para=[20 50 0.25 0.20 1]; end
if nargin<5, Ub=[]; end
if nargin<4, Lb=[]; end
if nargin<3,
disp('Usuage: FA_mincon(@cost,
@constraint,u0,Lb,Ub,para)');
end

% n=number of fireflies
% MaxGeneration=number of pseudo time steps
% -----------------------------------------------
% alpha=0.25;      % Randomness 0--1 (highly
random)
% betamn=0.20;     % minimum value of beta
% gamma=1;         % Absorption coefficient
% -----------------------------------------------
n=para(1);  MaxGeneration=para(2);
alpha=para(3); betamin=para(4); gamma=para(5);

% Total number of function evaluations
NumEval=n*MaxGeneration;

% Check if the upper bound & lower bound are the
same size
if length(Lb) ~=length(Ub),
    disp('Simple bounds/limits are improper!');
    return
end

% Calcualte dimension
d=length(u0);

% Initial values of an array
zn=ones(n,1)*10^100;
% -----------------------------------------------
% generating the initial locations of n fireflies
[ns,Lightn]=init_ffa(n,d,Lb,Ub,u0);
```

```
% Iterations or pseudo time marching
for k=1:MaxGeneration,     %%%%% start iterations

% This line of reducing alpha is optional
 alpha=alpha_new(alpha,MaxGeneration);

% Evaluate new solutions (for all n fireflies)
for i=1:n,
   zn(i)=Fun(fhandle,nonhandle,ns(i,:));
   Lightn(i)=zn(i);
end

% Ranking fireflies by their light
intensity/objectives
[Lightn,Index]=sort(zn);
ns_tmp=ns;
for i=1:n,
 ns(i,:)=ns_tmp(Index(i),:);
end

%% Find the current best
nso=ns; Lighto=Lightn;
nbest=ns(1,:); Lightbest=Lightn(1);

% For output only
fbest=Lightbest;

% Move all fireflies to the better locations
[ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,nbest,...
      Lightbest,alpha,betamin,gamma,Lb,Ub);

end   %%%%% end of iterations

% ----------------------------------------------
------
% ----- All the subfunctions are listed here ------
------
% The initial locations of n fireflies
function [ns,Lightn]=init_ffa(n,d,Lb,Ub,u0)
  % if there are bounds/limits,
if length(Lb)>0,
   for i=1:n,
```

```
   ns(i,:)=Lb+(Ub-Lb).*rand(1,d);
   end
else
   % generate solutions around the random guess
   for i=1:n,
   ns(i,:)=u0+randn(1,d);
   end
end

% initial value before function evaluations
Lightn=ones(n,1)*10^100;

% Move all fireflies toward brighter ones
function [ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,...

nbest,Lightbest,alpha,betamin,gamma,Lb,Ub)
% Scaling of the system
scale=abs(Ub-Lb);

% Updating fireflies
for i=1:n,
% The attractiveness parameter beta=exp(-gamma*r)
   for j=1:n,
      r=sqrt(sum((ns(i,:)-ns(j,:)).^2));
      % Update moves
if Lightn(i)>Lighto(j), % Brighter and more
attractive
   beta0=1; beta=(beta0-betamin)*exp(-
gamma*r.^2)+betamin;
   tmpf=alpha.*(rand(1,d)-0.5).*scale;
   ns(i,:)=ns(i,:).*(1-beta)+nso(j,:).*beta+tmpf;
      end
   end % end for j

end % end for i

% Check if the updated solutions/locations are
within limits
[ns]=findlimits(n,ns,Lb,Ub);

% This function is optional, as it is not in the
original FA
```

```
% The idea to reduce randomness is to increase the
convergence,
% however, if you reduce randomness too quickly,
then premature
% convergence can occur. So use with care.
function alpha=alpha_new(alpha,NGen)
% alpha_n=alpha_0(1-delta)^NGen=10^(-4);
% alpha_0=0.9
delta=1-(10^(-4)/0.9)^(1/NGen);
alpha=(1-delta)*alpha;

% Make sure the fireflies are within the
bounds/limits
function [ns]=findlimits(n,ns,Lb,Ub)
for i=1:n,
     % Apply the lower bound
  ns_tmp=ns(i,:);
  I=ns_tmp<Lb;
  ns_tmp(I)=Lb(I);

  % Apply the upper bounds
  J=ns_tmp>Ub;
  ns_tmp(J)=Ub(J);
  % Update this new move
  ns(i,:)=ns_tmp;
end

% ----------------------------------------
% d-dimensional objective function
function z=Fun(fhandle,nonhandle,u)
% Objective
z=fhandle(u);

% Apply nonlinear constraints by the penalty method
% Z=f+sum_k=1^N lam_k g_k^2 *H(g_k) where lam_k >>
1
z=z+getnonlinear(nonhandle,u);

function Z=getnonlinear(nonhandle,u)
Z=0;
% Penalty constant >> 1
lam=10^15; lameq=10^15;
```

```
% Get nonlinear constraints
[g,geq]=nonhandle(u);
% Apply inequality constraints as a penalty
function
for k=1:length(g),
    Z=Z+ lam*g(k)^2*getH(g(k));
end
% Apply equality constraints (when geq=[], length-
>0)
for k=1:length(geq),
   Z=Z+lameq*geq(k)^2*geteqH(geq(k));
end

% Test if inequalities hold
% H(g) which is something like an index function
function H=getH(g)
if g<=0,
    H=0;
else
    H=1;
end

% Test if equalities hold
function H=geteqH(g)
if g==0,
    H=0;
else
    H=1;
end
%% ==== End of Firefly Algorithm implementation
======
```

10.2. ANEXO B – Algoritmo Firefly adaptado para resolução dos problemas de variáveis contínuas

Essa rotina foi usada para resolução dos problemas 6.1, 6.2 e 7.1, alterando-se os seguintes valores, para se adaptarem ao problema:

lambda – Fator para aplicação da penalização quando uma das restrições não é respeitada.

para – Parâmetros do Algoritmo, na seguinte ordem: **Nº de Vagalumes**, **Nº de Iterações**, **α**, **β_0** e **γ**.

Lb e Ub – Limites Inferior e Superior, respectivamente, das variáveis.

```
function [u,fval]=fa_mincon
% parameters [n N_iteration alpha betamin gamma]
global lambda
lambda=100;
para=[20 300 0.5 0.5 1];
format long;

% Simple bounds/limits
Lb=[0  0];
Ub=[20  20];

% Initial random guess
u0=Lb+(Ub-Lb).*rand(size(Lb));

[u,fval,NumEval]=ffa_mincon(@cost,@constraint,u0,Lb
,Ub,para);

99;
%%% Put your own cost/objective function here -----
---%%%
%% Cost or Objective function
function L=cost(x)
global lambda

[f,g]=fobj(x);

P=max([g ;zeros(1,length(g))]);
%calcular o "Lagrangeano"
L=f + lambda*sum(P);

% Constrained optimization using penalty methods
% by changing f to F=f+ \sum lam_j*g^2_j*H_j(g_j)
% where H(g)=0 if g<=0 (true), =1 if g is false
```

```
%%% Put your own constraints here ----------------
---%%%
function [g,geq]=constraint(x)
% All nonlinear inequality constraints should be
here
% If no inequality constraint at all, simple use
g=[];
g=[];
% all nonlinear equality constraints should be here
% If no equality constraint at all, put geq=[] as
follows
geq=[];

%%% End of the part to be modified ---------------
---%%%

%%% ------------------------------------------------
---%%%
%%% Do not modify the following codes unless you
want %%%
%%% to improve its performance etc
%%%
% ------------------------------------------------
------
% ===Start of the Firefly Algorithm Implementation
======
% Inputs: fhandle => @cost (your own cost function,
%                   can be an external file  )
%     nonhandle => @constraint, all nonlinear
constraints
%                   can be an external file or a
function
%         Lb = lower bounds/limits
%         Ub = upper bounds/limits
%   para == optional (to control the Firefly
algorithm)
% Outputs: nbest   = the best solution found so far
%          fbest   = the best objective value
%      NumEval = number of evaluations:
n*MaxGeneration
% Optional:
```

```
% The alpha can be reduced (as to reduce the
randomness)
% ------------------------------------------------
--------

% Start FA
function [nbest,fbest,NumEval]...
           =ffa_mincon(fhandle,nonhandle,u0, Lb,
Ub, para)
% Check input parameters (otherwise set as default
values)
if nargin<6, para=[20 50 0.25 0.20 1]; end
if nargin<5, Ub=[]; end
if nargin<4, Lb=[]; end
if nargin<3,
disp('Usuage: FA_mincon(@cost,
@constraint,u0,Lb,Ub,para)');
end

% n=number of fireflies
% MaxGeneration=number of pseudo time steps
% ------------------------------------------------
% alpha=0.25;      % Randomness 0--1 (highly
random)
% betamn=0.20;     % minimum value of beta
% gamma=1;         % Absorption coefficient
% ------------------------------------------------
n=para(1);  MaxGeneration=para(2);
alpha=para(3); betamin=para(4); gamma=para(5);

% Total number of function evaluations
NumEval=n*MaxGeneration;

% Check if the upper bound & lower bound are the
same size
if length(Lb) ~=length(Ub),
    disp('Simple bounds/limits are improper!');
    return
end

% Calcualte dimension
d=length(u0);
```

```
% Initial values of an array
zn=ones(n,1)*10^100;
% ------------------------------------------------
% generating the initial locations of n fireflies
[ns,Lightn]=init_ffa(n,d,Lb,Ub,u0);

% Iterations or pseudo time marching
for k=1:MaxGeneration,     %%%%% start iterations

% This line of reducing alpha is optional
 alpha=alpha_new(alpha,MaxGeneration);

% Evaluate new solutions (for all n fireflies)
for i=1:n,
   zn(i)=Fun(fhandle,nonhandle,ns(i,:));
   Lightn(i)=zn(i);
end

% Ranking fireflies by their light
intensity/objectives
[Lightn,Index]=sort(zn);
ns_tmp=ns;
for i=1:n,
 ns(i,:)=ns_tmp(Index(i),:);
end

%% Find the current best
nso=ns; Lighto=Lightn;
nbest=ns(1,:); Lightbest=Lightn(1);

% For output only
fbest=Lightbest;

% Move all fireflies to the better locations
[ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,nbest,...
      Lightbest,alpha,betamin,gamma,Lb,Ub);

end   %%%%% end of iterations
```

```
% ------------------------------------------------
------
% ----- All the subfunctions are listed here ------
------
% The initial locations of n fireflies
function [ns,Lightn]=init_ffa(n,d,Lb,Ub,u0)
  % if there are bounds/limits,
if length(Lb)>0,
   for i=1:n,
   ns(i,:)=Lb+(Ub-Lb).*rand(1,d);
   end
else

   % generate solutions around the random guess
   for i=1:n,
   ns(i,:)=u0+randn(1,d);
   end
end

% initial value before function evaluations
Lightn=ones(n,1)*10^100;

% Move all fireflies toward brighter ones
function [ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,...

nbest,Lightbest,alpha,betamin,gamma,Lb,Ub)
% Scaling of the system
scale=abs(Ub-Lb);

% Updating fireflies
for i=1:n,
% The attractiveness parameter beta=exp(-gamma*r)
   for j=1:n,
      r=sqrt(sum((ns(i,:)-ns(j,:)).^2));
      % Update moves
if Lightn(i)>Lighto(j), % Brighter and more
attractive
   beta0=1; beta=(beta0-betamin)*exp(-
gamma*r.^2)+betamin;
   tmpf=alpha.*(rand(1,d)-0.5).*scale;
   ns(i,:)=ns(i,:).*(1-beta)+nso(j,:).*beta+tmpf;
      end
```

```
   end % end for j

end % end for i

% Check if the updated solutions/locations are
within limits
[ns]=findlimits(n,ns,Lb,Ub);

% This function is optional, as it is not in the
original FA
% The idea to reduce randomness is to increase the
convergence,
% however, if you reduce randomness too quickly,
then premature
% convergence can occur. So use with care.
function alpha=alpha_new(alpha,NGen)
% alpha_n=alpha_0(1-delta)^NGen=10^(-4);
% alpha_0=0.9
delta=1-(10^(-4)/0.9)^(1/NGen);
alpha=(1-delta)*alpha;

% Make sure the fireflies are within the
bounds/limits
function [ns]=findlimits(n,ns,Lb,Ub)
for i=1:n,
     % Apply the lower bound
  ns_tmp=ns(i,:);
  I=ns_tmp<Lb;
  ns_tmp(I)=Lb(I);

  % Apply the upper bounds
  J=ns_tmp>Ub;
  ns_tmp(J)=Ub(J);
  % Update this new move
  ns(i,:)=ns_tmp;
end

% ----------------------------------------
% d-dimensional objective function
function z=Fun(fhandle,nonhandle,u)

% Objective
```

```
z=fhandle(u);

% Apply nonlinear constraints by the penalty method
% Z=f+sum_k=1^N lam_k g_k^2 *H(g_k) where lam_k >>
1
z=z+getnonlinear(nonhandle,u);

function Z=getnonlinear(nonhandle,u)
Z=0;
% Penalty constant >> 1
lam=10^15; lameq=10^15;
% Get nonlinear constraints
[g,geq]=nonhandle(u);

% Apply inequality constraints as a penalty
function
for k=1:length(g),
    Z=Z+ lam*g(k)^2*getH(g(k));
end
% Apply equality constraints (when geq=[], length-
>0)
for k=1:length(geq),
   Z=Z+lameq*geq(k)^2*geteqH(geq(k));
end

% Test if inequalities hold
% H(g) which is something like an index function
function H=getH(g)
if g<=0,
    H=0;
else
    H=1;
end

% Test if equalities hold
function H=geteqH(g)
if g==0,
    H=0;
else
    H=1;
end
```

```
%% ==== End of Firefly Algorithm implementation
======
```

10.3. ANEXO C – Algoritmo Firefly adaptado para resolução dos problemas de variáveis mistas

Essa rotina foi usada para resolução dos problemas 7.2 e 7.3, alterando-se os seguintes valores, para se adaptarem ao problema:

lambda – Fator para aplicação da penalização quando uma das restrições não é respeitada.

para – Parâmetros do Algoritmo, na seguinte ordem: **Nº de Vagalumes**, **Nº de Iterações**, α, β_0 e γ.

perfis – Valores de área de seção disponíveis.

Lb e Ub – Limites Inferior e Superior, respectivamente, das variáveis.

```
%
================================================
======== %
% Files of the Matlab programs included in the
book:       %
% Xin-She Yang, Nature-Inspired Metaheuristic
Algorithms,  %
% Second Edition, Luniver Press, (2010).
www.luniver.com %
%
================================================
======== %

% ---------------------------------------------
---------- %
% Firefly Algorithm for constrained optimization
using     %
```

```
% for the design of a spring (benchmark)
%
% by Xin-She Yang (Cambridge University)
Copyright @2009   %
% ------------------------------------------
---------- %

function [u,fval]=fa_mincon
% parameters [n N_iteration alpha betamin gamma]
global lambda
lambda=25;
para=[10  800  1  .5   1];
format long;

% Simple bounds/limits

global perfis;
perfis=[.111 .141 .174 .220 .270 .287 .247 .440
.539 .954 1.081 1.174 1.333 1.488 1.764 2.142
2.697 2.800 3.131 3.565 3.813 4.805 5.952 6.572
7.192 8.525 9.300 10.85 13.33 14.29 17.17
19.18];
aux=length(perfis);
Lb=[.0001 .0001 .0001 .0001 .0001 .0001 .0001
.0001 .0001 .0001 .0001 .0001 .0001 .0001 .0001
100 220 100 100 50 -20 -20 20];
Ub=[aux   aux   aux   aux   aux   aux   aux
aux   aux   aux   aux   aux   aux   aux   aux
140 260 140 140 90 20  20  60];

% Initial random guess
u0=Lb+(Ub-Lb).*rand(size(Lb));

[u,NumEval]=ffa_mincon(@cost,@constraint,u0,Lb,U
b,para);
[peso,g,Mten]=trelica_2D(u);

for n=1:15
    D(n)=perfis(ceil(u(n)));
end
```

```
peso

99;
%%% Put your own cost/objective function here --
------%%%
%% Cost or Objective function
 function L=cost(A)
global lambda

[peso,g]=trelica_2D(A);

P=max([g ;zeros(1,length(g))]);
%calcular o "Lagrangeano"
L=peso + lambda*sum(P);
% Constrained optimization using penalty methods
% by changing f to F=f+ \sum
lam_j*g^2_j*H_j(g_j)
% where H(g)=0 if g<=0 (true), =1 if g is false

%%% Put your own constraints here --------------
------%%%
function [g,geq]=constraint(A)
% All nonlinear inequality constraints should be
here
% If no inequality constraint at all, simple use
g=[];
g=[];
% all nonlinear equality constraints should be
here
% If no equality constraint at all, put geq=[]
as follows
geq=[];

%%% End of the part to be modified -------------
------%%%

%%% -----------------------------------------
------%%%
```

```
%%% Do not modify the following codes unless you
want %%%
%%% to improve its performance etc
%%%
% ------------------------------------------------
---------
% ===Start of the Firefly Algorithm
Implementation ======
% Inputs: fhandle => @cost (your own cost
function,
%                   can be an external file  )
%     nonhandle => @constraint, all nonlinear
constraints
%                   can be an external file or a
function
%         Lb = lower bounds/limits
%         Ub = upper bounds/limits
%   para == optional (to control the Firefly
algorithm)
% Outputs: nbest   = the best solution found so
far
%          fbest   = the best objective value
%      NumEval = number of evaluations:
n*MaxGeneration
% Optional:
% The alpha can be reduced (as to reduce the
randomness)
% ------------------------------------------------
-----------

% Start FA
function [nbest,fbest,NumEval]...
           =ffa_mincon(fhandle,nonhandle,u0, Lb,
Ub, para)
% Check input parameters (otherwise set as
default values)
if nargin<6, para=[20 50 0.25 0.20 1]; end
if nargin<5, Ub=[]; end
if nargin<4, Lb=[]; end
if nargin<3,
```

```
disp('Usuage: FA_mincon(@cost,
@constraint,u0,Lb,Ub,para)');
end

% n=number of fireflies
% MaxGeneration=number of pseudo time steps
% ------------------------------------------------
--
% alpha=0.25;      % Randomness 0--1 (highly
random)
% betamn=0.20;     % minimum value of beta
% gamma=1;         % Absorption coefficient
% ------------------------------------------------
--
n=para(1);  MaxGeneration=para(2);
alpha=para(3); betamin=para(4); gamma=para(5);

% Total number of function evaluations
NumEval=n*MaxGeneration;

% Check if the upper bound & lower bound are the
same size
if length(Lb) ~=length(Ub),
    disp('Simple bounds/limits are improper!');
    return
end

% Calcualte dimension
d=length(u0);

% Initial values of an array
zn=ones(n,1)*10^100;
% ------------------------------------------------
--
% generating the initial locations of n
fireflies
[ns,Lightn]=init_ffa(n,d,Lb,Ub,u0);

% Iterations or pseudo time marching
```

```
for k=1:MaxGeneration,     %%%%% start
iterations

% This line of reducing alpha is optional
 alpha=alpha_new(alpha,MaxGeneration);

% Evaluate new solutions (for all n fireflies)
for i=1:n,
   zn(i)=Fun(fhandle,nonhandle,ns(i,:));
   Lightn(i)=zn(i);
end

% Ranking fireflies by their light
intensity/objectives
[Lightn,Index]=sort(zn);
ns_tmp=ns;
for i=1:n,
 ns(i,:)=ns_tmp(Index(i),:);
end

%% Find the current best
nso=ns; Lighto=Lightn;
nbest=ns(1,:); Lightbest=Lightn(1);

% For output only
fbest=Lightbest;

% Move all fireflies to the better locations
[ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,nbest,...
      Lightbest,alpha,betamin,gamma,Lb,Ub);

end   %%%%% end of iterations

% ----------------------------------------------
---------
% ----- All the subfunctions are listed here ---
---------
% The initial locations of n fireflies
function [ns,Lightn]=init_ffa(n,d,Lb,Ub,u0)
  % if there are bounds/limits,
```

```
if length(Lb)>0,
   for i=1:n,
   ns(i,:)=Lb+(Ub-Lb).*rand(1,d);
   end
else
   % generate solutions around the random guess
   for i=1:n,
   ns(i,:)=u0+randn(1,d);
   end
end

% initial value before function evaluations
Lightn=ones(n,1)*10^100;

% Move all fireflies toward brighter ones
function
[ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,...

nbest,Lightbest,alpha,betamin,gamma,Lb,Ub)
% Scaling of the system
scale=abs(Ub-Lb);
% Updating fireflies
for i=1:n,
% The attractiveness parameter beta=exp(-
gamma*r)
   for j=1:n,
      r=sqrt(sum((ns(i,:)-ns(j,:)).^2));
      % Update moves
if Lightn(i)>Lighto(j), % Brighter and more
attractive
   beta0=1; beta=(beta0-betamin)*exp(-
gamma*r.^2)+betamin;
   tmpf=alpha.*(rand(1,d)-0.5).*scale;
   ns(i,:)=ns(i,:).*(1-
beta)+nso(j,:).*beta+tmpf;
      end
   end % end for j
end % end for i

% Check if the updated solutions/locations are
within limits
```

```
[ns]=findlimits(n,ns,Lb,Ub);

% This function is optional, as it is not in the
original FA
% The idea to reduce randomness is to increase
the convergence,
% however, if you reduce randomness too quickly,
then premature
% convergence can occur. So use with care.
function alpha=alpha_new(alpha,NGen)
% alpha_n=alpha_0(1-delta)^NGen=10^(-4);
% alpha_0=0.9
delta=1-(10^(-4)/0.9)^(1/NGen);
alpha=(1-delta)*alpha;

% Make sure the fireflies are within the
bounds/limits
function [ns]=findlimits(n,ns,Lb,Ub)
for i=1:n,
     % Apply the lower bound
  ns_tmp=ns(i,:);
  I=ns_tmp<Lb;
  ns_tmp(I)=Lb(I);

  % Apply the upper bounds
  J=ns_tmp>Ub;
  ns_tmp(J)=Ub(J);
  % Update this new move
  ns(i,:)=ns_tmp;
end

% d-dimensional objective function
function z=Fun(fhandle,nonhandle,u)
% Objective
z=fhandle(u);
% Apply nonlinear constraints by the penalty
method
% Z=f+sum_k=1^N lam_k g_k^2 *H(g_k) where lam_k
>> 1
z=z+getnonlinear(nonhandle,u);
```

```
function Z=getnonlinear(nonhandle,u)
Z=0;
% Penalty constant >> 1
lam=10^15; lameq=10^15;
% Get nonlinear constraints
[g,geq]=nonhandle(u);

% Apply inequality constraints as a penalty
function
for k=1:length(g),
    Z=Z+ lam*g(k)^2*getH(g(k));
end
% Apply equality constraints (when geq=[],
length->0)
for k=1:length(geq),
   Z=Z+lameq*geq(k)^2*geteqH(geq(k));
end

% Test if inequalities hold
% H(g) which is something like an index function
function H=getH(g)
if g<=0,
    H=0;
else
    H=1;
end

% Test if equalities hold
function H=geteqH(g)
if g==0,
    H=0;
else
    H=1;
end
%% ==== End of Firefly Algorithm implementation
=====
```

10.4. ANEXO D – Funções Objetivos dos problemas apresentados

10.4.1. Função do 2ª Grau (6.1)

```
function [f,g]=fobj(x)

% Função objetivo
f=x(1)^2+x(2)^2-2*x(1)-2*x(2);
% Restrições de desigualdade
g(1)= x(1)+x(2)-4;
g(2)= 2-x(1);
```

10.4.2. Viga Metálica (6.2)

```
function [vol,g]=fobj(x)

% Parâmentos:
L=25;          %Comprimento (m)
Sd=262;        %Tensão de Escoamento (MPa)
Sma=0.55*Sd;   %Tensão de Flexão Admissível (MPa)
Sfa=255;       %Tensão de Fadiga Admissível (MPa)
Ta=0.33*Sd;    %Tensão de Cisalhamento Admissível
(MPa)
fla=L/800;     %Flecha Admissível (m)
Pm=104;        %Carga Concentrada para Momento
(kN)
Pv=155;        %Carga Concentrada para Cortante
(kN)
E=210;         %Módulo de Elasticidade (GPa)

% Variáveis de Projeto:
% x(1): Altura da alma(m)      - h
% x(2): Espessura da Alma (m) - tw
% x(3): Largura da Mesa (m)    - b
% x(4): Espessura da Mesa (m) - tf

% Variáveis Dependentes:
```

```
A=x(1)*x(2) + 2*x(3)*x(4);                  %Área da
Seção (m²)
I=(x(2)*(x(1)^3))/12 + 2*(x(3)*(x(4)^3))/3 +
x(3)*x(4)*x(1)*(x(1) + 2*x(4))/2;
%Momento de Inércia (m^4)
q=19 + 77*A;                                %Carga
Distribuída (kN/m)
M=L/8*(2*Pm + q*L);                         %Momento
Fletor (kN*m)
Sm=M*(x(1)/2 + x(4))/(1000*I);              %Tensão
de Flexão (MPa)
Sf=72845*(x(4)/x(3))^2;                     %Tensão
de Flambagem da Mesa (MPa)
Sw=3648276*(x(2)/x(1))^2;                   %Tensão
de Flambagem da Alma (MPa)
V=(Pv + q*25)/2;                            %Esforço
Cortante (kN)
fl=(8*Pm + 5*q*L)*(L^3)/((384e6)*E*I);      %Flecha
(m)
T=V/(1000*x(1)*x(2));                       %Tensão
de Cisalhamento (MPa)

% Volume:
vol=(x(1)*x(2) + 2*x(3)*x(4))*L;


% Restrições
g(1)=Sm/Sma - 1;
g(2)=Sm/Sf - 1;
g(3)=Sm/Sw - 1;
g(4)=T/Ta - 1;
g(5)=fl/fla - 1;
g(6)=2*Sm/Sfa - 1;
```

10.4.3. Treliça com 10 Barras (7.1)

```
function [peso,g]=fobj(A)
% clear;
% clc;
```

```
% Propriedades
E=1.0e7;            %Módulo de Elasticidade
(psi)
tenadm=2.5e4;       %Tensão Adm    (psi)
dadm=2;             %Desloc. Adm   (pol)
dens=.1;            %Densidade     (lb/pol³)
% Número de nós e elementos
n_nos=6;
n_el=10;
% número de cada nó, coordenada x e y
no = [1     2     3     4     5     6];
x  = [720   720   360   360   0     0];
y  = [360   0     360   0     360   0];

% Conectividade: [elemento   seção    nó_1
nó_2]
conec = [1   1   5   3
         2   2   3   1
         3   3   6   4
         4   4   4   2
         5   5   4   3
         6   6   2   1
         7   7   5   4
         8   8   6   3
         9   9   3   2
         10  10  4   1];

% Seções: [número da seção  área  módulo de
elasticidade]
n_sec=10;
secoes = [1   A(1)   E
          2   A(2)   E
          3   A(3)   E
          4   A(4)   E
          5   A(5)   E
          6   A(6)   E
          7   A(7)   E
          8   A(8)   E
          9   A(9)   E
          10  A(10)  E];
```

```
% Carregamentos, forças=[nó   intensidade_x
intensidade_y]
n_forcas=2;
forcas=[2   0   -1.0e5
        4   0   -1.0e5];

% Apoios
n_rest=2;  %número de nós restringidos
GDL_rest=[5   1   1
          6   1   1];  %[nó   restringido_x
restringido_y] (1 para restringido, e 0 para
livre)

% CALCULO DA ESTRUTURA
GDL=2*n_nos;  %graus de liberdade da estrutura
K=zeros(GDL,GDL); %matriz rigidez global

% Cálculo da matriz de cada elemento
for el=1:n_el
    %calculo do comprimento do elemento el
    no1=conec(el,3);
    no2=conec(el,4);
    %L=abs(x(no2)-x(no1));
    L(el) = sqrt((x(no2) - x(no1))^2 + (y(no2) -
y(no1))^2);
    %Propriedades
    A(el) = secoes(conec(el,2),2);
    E(el) = secoes(conec(el,2),3);
    %Cossenos diretores a partir das coordenadas
dos nós do elemento
    cs = (x(no2) - x(no1))/L(el);    % cosseno
    sn = (y(no2) - y(no1))/L(el);    %  seno
    % Matriz de rigidez do elemento "el"
    k=E(el)*A(el)/L(el);
    ke=[k  -k
       -k   k];
    T=[cs  sn   0   0
        0   0  cs  sn];
    kg=T'*ke*T;
    %Inserção na matriz de rigidez global
```

```
    for i=1:2                              %
superposição da sub-matriz (1-2,1-2) da matriz
elementar
      ig = (no1-1)*2+i;
      for j=1:2
         jg = (no1-1)*2+j;
         K(ig,jg)=K(ig,jg)+kg(i,j);
      end
   end

   for i=1:2                 % superposição da
sub-matriz (3-4,3-4) da matriz elementar
       ig = (no2-1)*2+i;
       for j=1:2
           jg = (no2-1)*2+j;
           K(ig,jg)=K(ig,jg)+kg(i+2,j+2);
       end
   end

   for i=1:2                 % superposição das
sub-matrizes (1-2,3-4) e ((3-4,1-2) da matriz
elementar
       ig = (no1-1)*2+i;
       for j=1:2
           jg = (no2-1)*2+j;
           K(ig,jg)=K(ig,jg)+kg(i,j+2);
           K(jg,ig)=K(jg,ig)+kg(j+2,i);
       end
   end

end

% Vetor de forças Global
F=zeros(GDL,1);
for i=1:n_forcas
    F(2*forcas(i,1)-1)=forcas(i,2);
    F(2*forcas(i,1))=forcas(i,3);
end
% guardamos os originais de K e F
```

```
Kg=K;
Fg=F;
% Aplicar Restrições (condições de contorno)
for k=1:n_rest
    % Verifica se há restrição na direção x
    if GDL_rest(k,2)==1
        j=2*GDL_rest(k,1)-1;
        %Modificar Matriz de Rigidez
        for i=1:GDL
            Kg(j,i)=0;    %zera linha
            Kg(i,j)=0;    %zera coluna
        end
        Kg(j,j)=1;        %valor unitário na
dianogal principal
        Fg(j)=0;
    end
    % Verifica se há restrição na direção y
    if GDL_rest(k,3)==1
        j=2*GDL_rest(k,1);
        %Modificar Matriz de Rigidez
        for i=1:GDL
            Kg(j,i)=0;    %zera linha
            Kg(i,j)=0;    %zera coluna
        end
        Kg(j,j)=1;        %valor unitário na
dianogal principal
        Fg(j)=0;
    end
end

%Mostrando K e F na tela:
% disp('Matriz de rigidez global:')
% disp(K)
% disp('Vetor de forças globais:')
% disp(F)


% Calculo dos deslocamentos
desloc=Kg\Fg;

% disp('Deslocamentos obtidos:')
```

```
% disp(desloc)

% Reações
reacoes=K*desloc;

% Esforçoes nos elementos
for el=1:n_el
    %calculo do comprimento do elemento "el"
    no1=conec(el,3);
    no2=conec(el,4);
    L(el) = sqrt((x(no2) - x(no1))^2 + (y(no2) -
y(no1))^2);
    %Propriedades
    A(el) = secoes(conec(el,2),2);
    E(el) = secoes(conec(el,2),3);
    %Cossenos diretores a partir das coordenadas
dos nós do elemento
    cs = (x(no2) - x(no1))/L(el);    % cosseno
    sn = (y(no2) - y(no1))/L(el);    %  seno
    %pega os valores dos deslocamentos dos nós
do elemento "el"
    u1 = desloc(no1*2-1);
    u2 = desloc(no2*2-1);
    v1 = desloc(no1*2);
    v2 = desloc(no2*2);
    %constante de rigidez do elemento "el"
    k=E(el)*A(el)/L(el);
    %força e tensão atuante no elemento "el"
    fn(el) = k*(-(u1-u2)*cs - (v1-v2)*sn);
% cálculo da força normal no elemento
    ten(el) = fn(el)/A(el);        % cálculo da
tensão normal no elemento
end

%Módulo das Tensões
for el=1:n_el
Mten(el)=((ten(el)*ten(el))^0.5);
g(el)=Mten(el)/tenadm - 1;
end

%Módulo dos deslocamentos
```

```
for n=1:2*n_nos
Mdesloc(n)=((desloc(n)*desloc(n))^0.5);
g(n_el+n)=Mdesloc(n)/dadm - 1;
end


%Peso de cada barra
for el=1:n_el
    pesobarra(el)=dens*A(el)*L(el);
end

%Peso Total
peso=sum(pesobarra);
```

10.4.4. Treliça com 15 Barras (7.2)

Aqui é apresentada a rotina usada para o cálculo do caso 2, onde é considerada a restrição de flambagem:

```
function [peso,g,Mten]=trelica_2D(A)

% Propriedades
E=1e7;              %Módulo de Elasticidade  (psi)
tenadm=25e3;        %Tensão Adm   (psi)
dens=.1;            %Densidade    (lb/pol³)

global perfis;
for n=1:15
    D(n)=perfis(ceil(A(n)));
end
for n=16:23
    D(n)=A(n);
end

A=D;

% Número de nós e elementos
n_nos=8;
n_el=15;
```

```
% Número de cada nó, coordenada x e y
no = [1     2     3     4     5     6     7
8];
x  = [0     A(16) A(17) 360   0     A(16) A(17)
360];
y  = [120   A(18) A(19) A(20) 0     A(21) A(22)
A(23)];

% Conectividade: [elemento   seção    nó_1
nó_2]
conec = [1   1   1   2
         2   2   2   3
         3   3   3   4
         4   4   5   6
         5   5   6   7
         6   6   7   8
         7   7   6   2
         8   8   7   3
         9   9   8   4
         10  10  1   6
         11  11  5   2
         12  12  2   7
         13  13  6   3
         14  14  3   8
         15  15  7   4];

% Seções: [número da seção  área  módulo de
elasticidade]
n_sec=15;
secoes = [1   A(1)   E
          2   A(2)   E
          3   A(3)   E
          4   A(4)   E
          5   A(5)   E
          6   A(6)   E
          7   A(7)   E
          8   A(8)   E
          9   A(9)   E
          10  A(10)  E
          11  A(11)  E
          12  A(12)  E
```

```
          13  A(13)  E
          14  A(14)  E
          15  A(15)  E];

% Carregamentos, forças=[nó   intensidade_x
intensidade_y]
n_forcas=1;
forcas=[8   0   -1e4];

% Apoios
n_rest=2;  %número de nós restringidos
GDL_rest=[1   1   1
          5   1   1];  %[nó   restringido_x
restringido_y] (1 para restringido, e 0 para
livre)

% CALCULO DA ESTRUTURA
GDL=2*n_nos;  %graus de liberdade da estrutura
K=zeros(GDL,GDL); %matriz rigidez global

% Cálculo da matriz de cada elemento
for el=1:n_el
    %calculo do comprimento do elemento el
    no1=conec(el,3);
    no2=conec(el,4);
    %L=abs(x(no2)-x(no1));
    L(el) = sqrt((x(no2) - x(no1))^2 + (y(no2) -
y(no1))^2);
    %Propriedades
    A(el) = secoes(conec(el,2),2);
    E(el) = secoes(conec(el,2),3);

    %Cossenos diretores a partir das coordenadas
dos nós do elemento
    cs = (x(no2) - x(no1))/L(el);    % cosseno
    sn = (y(no2) - y(no1))/L(el);    %  seno
    % Matriz de rigidez do elemento "el"
    k=E(el)*A(el)/L(el);
    ke=[k  -k
       -k   k];
```

```
    T=[cs  sn   0   0
       0   0  cs  sn];
    kg=T'*ke*T;
    %Inserção na matriz de rigidez global
    for i=1:2   % superposição da sub-matriz (1-
2,1-2) da matriz elementar
      ig = (no1-1)*2+i;
      for j=1:2
         jg = (no1-1)*2+j;
         K(ig,jg)=K(ig,jg)+kg(i,j);
      end
   end

   for i=1:2   % superposição da sub-matriz (3-
4,3-4) da matriz elementar
       ig = (no2-1)*2+i;
       for j=1:2
           jg = (no2-1)*2+j;
           K(ig,jg)=K(ig,jg)+kg(i+2,j+2);
       end
   end

   for i=1:2        % superposição das sub-
matrizes (1-2,3-4) e ((3-4,1-2) da matriz
elementar
       ig = (no1-1)*2+i;
       for j=1:2
           jg = (no2-1)*2+j;
           K(ig,jg)=K(ig,jg)+kg(i,j+2);
           K(jg,ig)=K(jg,ig)+kg(j+2,i);
       end
   end

end

% Vetor de forças Global
F=zeros(GDL,1);
for i=1:n_forcas
    F(2*forcas(i,1)-1)=forcas(i,2);
    F(2*forcas(i,1))=forcas(i,3);
end
```

```
% guardamos os originais de K e F
Kg=K;
Fg=F;
% Aplicar Restrições (condições de contorno)
for k=1:n_rest
    % Verifica se há restrição na direção x
    if GDL_rest(k,2)==1
        j=2*GDL_rest(k,1)-1;
        %Modificar Matriz de Rigidez
        for i=1:GDL
            Kg(j,i)=0;    %zera linha
            Kg(i,j)=0;    %zera coluna
        end
        Kg(j,j)=1;        %valor unitário na
dianogal principal
        Fg(j)=0;
    end
    % Verifica se há restrição na direção y
    if GDL_rest(k,3)==1
        j=2*GDL_rest(k,1);
        %Modificar Matriz de Rigidez
        for i=1:GDL
            Kg(j,i)=0;    %zera linha
            Kg(i,j)=0;    %zera coluna
        end
        Kg(j,j)=1;        %valor unitário na
dianogal principal
        Fg(j)=0;
    end
end

% Deslocamentos
desloc=Kg\Fg;

% Reações
reacoes=K*desloc;

% Esforçoes nos elementos
for el=1:n_el
    %calculo do comprimento do elemento "el"
    no1=conec(el,3);
```

```
    no2=conec(el,4);
    L(el) = sqrt((x(no2) - x(no1))^2 + (y(no2) -
y(no1))^2);
    %Propriedades
    A(el) = secoes(conec(el,2),2);
    E(el) = secoes(conec(el,2),3);
    %Cossenos diretores a partir das coordenadas
dos nós do elemento
    cs = (x(no2) - x(no1))/L(el);    % cosseno
    sn = (y(no2) - y(no1))/L(el);    %  seno
    %pega os valores dos deslocamentos dos nós
do elemento "el"
    u1 = desloc(no1*2-1);
    u2 = desloc(no2*2-1);
    v1 = desloc(no1*2);
    v2 = desloc(no2*2);
    %constante de rigidez do elemento "el"
    k=E(el)*A(el)/L(el);
    %força e tensão atuante no elemento "el"
    fn(el) = k*(-(u1-u2)*cs - (v1-v2)*sn);
% cálculo da força normal no elemento
    ten(el) = fn(el)/A(el);        % cálculo da
tensão normal no elemento
end

%Módulo das Tensões
for el=1:n_el
Mten(el)=((ten(el)*ten(el))^0.5);
g(el)=Mten(el)/tenadm - 1;
end


%Flambagem (apenas para o Caso 2)
for el=1:n_el
FCrit(el)=100*E(el)*A(el)/(8*(L(el)^2));
g(n_el+el)=-ten(el)/FCrit(el) - 1;
end

%Peso de cada barra
for el=1:n_el
    pesobarra(el)=dens*A(el)*L(el);
```

```
end

%Peso Total
peso=sum(pesobarra);
```

10.4.5. Treliça com 21 Barras (7.3)

```
function [peso,g]=trelica_2D(A)
% clear;
% clc;

% Propriedades

E=189e9;            %Módulo de Elasticidade   (N/m²)
tenadm=189e6;       %Tensão Adm   (N/m²)
dadm=.01;           %Desloc. Adm  (m)
dens=7.842e4;       %Densidade    (N/m³)

global perfis;
for n=1:21
    D(n)=perfis(ceil(A(n)));
end
for n=22:25
    D(n)=A(n);
end

A=D;

% Número de nós e elementos
n_nos=10;
n_el=21;

% número de cada nó, coordenada x e y
no = [1  2     3     4     5     6   7         8
9         10];
x  = [0  A(22) A(23) A(24) A(25) 20  20-A(22) 20-
A(23) 20-A(24) 20-A(25)];
y  = [0  10    20    30    40    0   10        20
30        40];
```

```
% Conectividade: [elemento   seção    nó_1    nó_2]
conec = [1   1   1   2
         2   2   2   3
         3   3   3   4
         4   4   4   5
         5   5   6   7
         6   6   7   8
         7   7   8   9
         8   8   9   10
         9   9   1   7
         10  10  2   6
         11  11  2   8
         12  12  3   7
         13  13  3   9
         14  14  4   8
         15  15  4   10
         16  16  5   9
         17  17  1   6
         18  18  2   7
         19  19  3   8
         20  20  4   9
         21  21  5   10];

% Seções: [número da seção  área  módulo de
elasticidade]
n_sec=21;
secoes = [1   .95*A(1)   E
          2   .95*A(2)   E
          3   .95*A(3)   E
          4   .95*A(4)   E
          5   .95*A(5)   E
          6   .95*A(6)   E
          7   .95*A(7)   E
          8   .95*A(8)   E
          9   .95*A(9)   E
          10  .95*A(10)  E
          11  .95*A(11)  E
          12  .95*A(12)  E
          13  .95*A(13)  E
          14  .95*A(14)  E
          15  .95*A(15)  E
          16  .95*A(16)  E
```

```
          17   .95*A(17)   E
          18   .95*A(18)   E
          19   .95*A(19)   E
          20   .95*A(20)   E
          21   .95*A(21)   E];

% Carregamentos, forças=[nó   intensidade_x
intensidade_y]
n_forcas1=2;
forcas1=[5    0    -1.1e5
         10   0    -1.1e5];

n_forcas2=5;
forcas2=[2    3.25e4   0
         3    3.25e4   0
         4    3.25e4   0
         5    3.25e4   -1.1e5
         10   0        -1.1e5];

n_forcas3=5;
forcas3=[5    0         -1.1e5
         7    -3.25e4   0
         8    -3.25e4   0
         9    -3.25e4   0
         10   -3.25e4   -1.1e5];

% Apoios
n_rest=2;  %número de nós restringidos
GDL_rest=[1   1   1
          6   1   1];  %[nó   restringido_x
restringido_y] (1 para restringido, e 0 para livre)

% CALCULO DA ESTRUTURA
GDL=2*n_nos;  %graus de liberdade da estrutura
K=zeros(GDL,GDL); %matriz rigidez global

% Cálculo da matriz de cada elemento
for el=1:n_el
    %calculo do comprimento do elemento el
    no1=conec(el,3);
    no2=conec(el,4);
    %L=abs(x(no2)-x(no1));
```

```
    L(el) = sqrt((x(no2) - x(no1))^2 + (y(no2) -
y(no1))^2);
    %Propriedades
    A(el) = secoes(conec(el,2),2);
    E(el) = secoes(conec(el,2),3);
    %Cossenos diretores a partir das coordenadas
dos nós do elemento
    cs = (x(no2) - x(no1))/L(el);    % cosseno
    sn = (y(no2) - y(no1))/L(el);    %  seno
    % Matriz de rigidez do elemento "el"
    k=E(el)*A(el)/L(el);
    ke=[k  -k
       -k   k];
    T=[cs  sn   0   0
        0   0  cs  sn];
    kg=T'*ke*T;
    %Inserção na matriz de rigidez global

    for i=1:2                          % superposição
da sub-matriz (1-2,1-2) da matriz elementar
      ig = (no1-1)*2+i;
      for j=1:2
         jg = (no1-1)*2+j;
         K(ig,jg)=K(ig,jg)+kg(i,j);
      end
   end

   for i=1:2                % superposição da sub-
matriz (3-4,3-4) da matriz elementar
       ig = (no2-1)*2+i;
       for j=1:2
           jg = (no2-1)*2+j;
           K(ig,jg)=K(ig,jg)+kg(i+2,j+2);
       end
   end

   for i=1:2                % superposição das sub-
matrizes (1-2,3-4) e ((3-4,1-2) da matriz elementar
       ig = (no1-1)*2+i;
       for j=1:2
           jg = (no2-1)*2+j;
           K(ig,jg)=K(ig,jg)+kg(i,j+2);
           K(jg,ig)=K(jg,ig)+kg(j+2,i);
```

```
        end
    end

end

% Vetor de forças Global
F1=zeros(GDL,1);
F2=zeros(GDL,1);
F3=zeros(GDL,1);
for i=1:n_forcas1
    F1(2*forcas1(i,1)-1)=forcas1(i,2);
    F1(2*forcas1(i,1))=forcas1(i,3);
end
for i=1:n_forcas2
    F2(2*forcas2(i,1)-1)=forcas2(i,2);
    F2(2*forcas2(i,1))=forcas2(i,3);
end
for i=1:n_forcas3
    F3(2*forcas3(i,1)-1)=forcas3(i,2);
    F3(2*forcas3(i,1))=forcas3(i,3);
end
% guardamos os originais de K e F
Kg=K;
Fg1=F1;
Fg2=F2;
Fg3=F3;
% Aplicar Restrições (condições de contorno)
for k=1:n_rest
    % Verifica se há restrição na direção x
    if GDL_rest(k,2)==1
        j=2*GDL_rest(k,1)-1;
        %Modificar Matriz de Rigidez
        for i=1:GDL
            Kg(j,i)=0;    %zera linha
            Kg(i,j)=0;    %zera coluna
        end
        Kg(j,j)=1;        %valor unitário na
dianogal principal
        Fg1(j)=0;
        Fg2(j)=0;
        Fg3(j)=0;
    end
    % Verifica se há restrição na direção y
```

```
    if GDL_rest(k,3)==1
        j=2*GDL_rest(k,1);
        %Modificar Matriz de Rigidez
        for i=1:GDL
            Kg(j,i)=0;    %zera linha
            Kg(i,j)=0;    %zera coluna
        end
        Kg(j,j)=1;        %valor unitário na
dianogal principal
        Fg1(j)=0;
        Fg2(j)=0;
        Fg3(j)=0;
    end
end

% Calculo dos deslocamentos
desloc1=Kg\Fg1;
desloc2=Kg\Fg2;
desloc3=Kg\Fg3;
% Reações
reacoes1=K*desloc1;
reacoes2=K*desloc2;
reacoes3=K*desloc3;

% Esforçoes nos elementos
for el=1:n_el
    %calculo do comprimento do elemento "el"
    no1=conec(el,3);
    no2=conec(el,4);
    L(el) = sqrt((x(no2) - x(no1))^2 + (y(no2) -
y(no1))^2);
    %Propriedades
    A(el) = secoes(conec(el,2),2);
    E(el) = secoes(conec(el,2),3);
    %Cossenos diretores a partir das coordenadas
dos nós do elemento
    cs = (x(no2) - x(no1))/L(el);    % cosseno
    sn = (y(no2) - y(no1))/L(el);    %  seno
    %pega os valores dos deslocamentos dos nós do
elemento "el"
    u11 = desloc1(no1*2-1);
    u21 = desloc1(no2*2-1);
    u12 = desloc2(no1*2-1);
```

```
    u22 = desloc2(no2*2-1);
    u13 = desloc3(no1*2-1);
    u23 = desloc3(no2*2-1);
    v11 = desloc1(no1*2);
    v21 = desloc1(no2*2);
    v12 = desloc2(no1*2);
    v22 = desloc2(no2*2);
    v13 = desloc3(no1*2);
    v23 = desloc3(no2*2);
    %constante de rigidez do elemento "el"
    k=E(el)*A(el)/L(el);
    %força e tensão atuante no elemento "el"
    fn1(el) = k*(-(u11-u21)*cs - (v11-v21)*sn);
% cálculo da força normal no elemento
    fn2(el) = k*(-(u12-u22)*cs - (v12-v22)*sn);
    fn3(el) = k*(-(u13-u23)*cs - (v13-v23)*sn);
    ten1(el) = fn1(el)/A(el);        % cálculo da
tensão normal no elemento
    ten2(el) = fn2(el)/A(el);
    ten3(el) = fn3(el)/A(el);
end

%Módulo das Tensões
for el=1:n_el
Mten1(el)=((ten1(el)*ten1(el))^0.5);
Mten2(el)=((ten2(el)*ten2(el))^0.5);
Mten3(el)=((ten3(el)*ten3(el))^0.5);
g(el)=Mten1(el)/tenadm - 1;
g(n_el+el)=Mten2(el)/tenadm - 1;
g(2*n_el+el)=Mten3(el)/tenadm - 1;
end

%Módulo dos deslocamentos
for n=1:2*n_nos
Mdesloc1(n)=((desloc1(n)*desloc1(n))^0.5);
Mdesloc2(n)=((desloc2(n)*desloc2(n))^0.5);
Mdesloc3(n)=((desloc3(n)*desloc3(n))^0.5);
g(3*n_el+n)=Mdesloc1(n)/dadm - 1;
g(3*n_el+2*n_nos+n)=Mdesloc2(n)/dadm - 1;
g(3*n_el+4*n_nos+n)=Mdesloc3(n)/dadm - 1;
end
```

```
%Flambagem
for el=1:n_el
Flim(el)=(pi^2)*E(el)*[.36*((.95*A(el))^2)]/((L(el)
^2));
g(3*n_el+6*n_nos+el)=-fn1(el)/Flim(el) - 1;
g(3*n_el+6*n_nos+n_el+el)=-fn2(el)/Flim(el) - 1;
g(3*n_el+6*n_nos+2*n_el+el)=-fn3(el)/Flim(el) - 1;
end

%Peso de cada barra
for el=1:n_el
    pesobarra(el)=dens*D(el)*L(el);
end

%Peso Total
peso=sum(pesobarra);
```