**Aplicação de Métodos Estocásticos para a Otimização Do Problema Apresentado**

**CEFET-RJ - Unidade de Nova Friburgo**

Disciplina: Problemas Inversos em Python

Professora: Josiele da Silva Teixeira

Aluno: Raul Martins Furtado Fernandes (raul.fernandes@aluno.cefet-rj.br)

**RESUMO.** O objetivo deste trabalho foi a implementação computacional de dois métodos estocásticos para determinar o valor ótimo do calor específico de uma placa de alumínio a partir de dados experimentais apresentados: Luus-Jaakola e Evolução Diferencial. São apresentados e analisados os resultados obtidos com cada método com diferentes combinações de parâmetros.

# INTRODUÇÃO

A otimização de funções é uma área fundamental na pesquisa operacional e na ciência da computação, desempenhando um papel crucial em diversos campos como engenharia, economia, biologia, entre outros. A capacidade de encontrar soluções ótimas para problemas complexos pode significar grandes avanços em eficiência e desempenho.

Métodos estocásticos de otimização têm sido amplamente estudados e aplicados devido à sua habilidade de lidar com problemas de otimização contínuos e não-lineares. Esses métodos, muitas vezes baseados em processos probabilísticos e aleatórios, oferecem alternativas robustas aos métodos clássicos de otimização. Neste trabalho, focamos em dois métodos: Luus-Jaakola e Evolução Diferencial, ambos com características únicas que os tornam adequados para diferentes tipos de problemas de otimização.

# APRESENTAÇÃO DO PROBLEMA E DOS MÉTODOS DE OTIMIZAÇÃO UTILIZADOS

A partir dos dados experimentais fornecidos, foi implementada uma função objetivo que calcula o custo associado ao calor específico da placa de alumínio (*cp*). O custo reflete a discrepância entre os valores de temperatura calculados pela função teórica e os valores observados experimentalmente para diferentes instantes de tempo (*t*). Assim, a função objetivo busca quantificar a precisão do *cp* proposto, permitindo a identificação do valor mais adequado por meio de métodos de otimização.

# Método Luus-Jaakola

O método de Luus-Jaakola é um algoritmo de busca estocástica que se baseia na geração de novas soluções dentro de um intervalo decrescente ao longo das iterações. Ele é apreciado por sua capacidade de encontrar soluções próximas do ótimo global com um consumo reduzido de recursos computacionais.

Este método recebe como parâmetro a função objetivo a ser otimizada (*fObj*), os limites mínimo e máximo para cada parâmetro de *fObj* (*bounds*), o número de iterações internas (*nInt*), o número de iterações externas (*nOut*) e o coeficiente de contração (*coef*).

O método Luus-Jaakola é valorizado pela sua simplicidade e eficiência, proporcionando um bom equilíbrio entre resultado e custo computacional. A parametrização adequada do número de iterações e do coeficiente de contração é essencial para obter bons resultados.

# Descrição do Algoritmo

1. Inicialização
   1. A variável *stepSize* é definida como a diferença entre o limite máximo e mínimo para cada parâmetro de *fObj*;
   2. Um valor aleatório (*x*) é gerado para cada parâmetro de *fObj* dentro dos limites estabelecidos;
   3. O resultado função objetivo para *x* é salvo em *fx*;
2. Iterações
   1. O algoritmo realiza *nOut* iterações externas. Em cada iteração externa são realizadas *nInt* iterações internas e em seguida *stepSize* é multiplicado por (1 – *coef*) para reduzir o intervalo de busca;
   2. Em cada iteração interna, uma nova solução (*newX*) é gerada adicionando uma variação aleatória proporcional ao *stepSize* à solução atual (*x*). Esta nova solução é ajustada para ficar dentro dos limites definidos e, em seguida, *fObj* é avaliada com *newX* e o resultado é salvo em *newFx*. Se *newFx* for menor que *fx*, a solução ótima atual *x* e seu valor *fx* são atualizados;
3. Fim
   1. O melhor valor encontrado para o parâmetro de *fObj* e assim como o resultado da função objetivo com esse valor são retornados.

# Método Evolução Diferencial

A Evolução Diferencial é um algoritmo estocástico de otimização que opera com uma população de soluções candidatas, utilizando operações de mutação, recombinação e seleção para explorar o espaço de busca e encontrar soluções ótimas.

Este método recebe como parâmetro a função objetivo a ser otimizada (*fObj*), os limites mínimo e máximo para o parâmetro de *fObj* (*bounds*), o tamanho da população (*nPop*), a quantidade de gerações (*nGen*), a taxa de mutação (*F*) e a taxa de recombinação (*CR*).

A Evolução Diferencial é valorizada por sua capacidade de encontrar soluções precisas através da combinação de indivíduos da população, promovendo a diversidade e explorando eficazmente o espaço de soluções. A correta escolha dos parâmetros é crucial para maximizar a performance e a eficiência do algoritmo, garantindo que ele possa explorar o espaço de busca de maneira equilibrada.

# Descrição do Algoritmo

1. Inicialização
   1. A variável *pop* é criada como um vetor de *nPop* elementos e cada elemento é gerado aleatoriamente obedecendo os limites mínimo e máximo *bounds*;
   2. A variável *results* é criada como um vetor de *nPop* elementos onde *results[i] = fObj(pop[i])*;
2. Gerações
   1. O algoritmo realiza *nGen* iterações que representam gerações. A cada geração são performadas operações de mutação, recombinação e seleção;
   2. Mutação
      1. A variável *mutantPop* é criada como um vetor de nPop elementos;
      2. São realizadas *nPop* iterações incrementando um contador *i*;
      3. A cada iteração são gerados 3 outros índices aleatórios diferentes de *i* e diferentes entre si;
      4. Um elemento mutante (*mutant*) é gerado a partir da seguinte fórmula: *pop[r1] + F \* (pop[r2] – pop[r3])*;
      5. *mutant* é ajustado para ficar dentro dos limites de *bounds* e adicionado no vetor *mutantPop;*
   3. Recombinação
      1. A variável *trialPop* é criada como uma cópia de *pop*;
      2. Um índice de *pop* é gerado aleatoriamente (*randI*);
      3. São realizadas *nPop* iterações incrementando um contador *i*;
      4. Em cada iteração, um valor aleatório entre 0 e 1 é gerado. Se esse valor for menor ou igual a *CR* ou se *i* for igual a *randI*, *trialPop[i] = mutantPop[i]*;
   4. Seleção
      1. O vetor *trialResults* é inicializado com o resultado da função objetivo para cada elemento de *trialPop*;
      2. Cada elemento de *trialResults* é comparado com o elemento de mesmo índice de *results*. Se for menor, os elementos de *pop* e de *results* dessa posição serão substituídos pelos elementos de *trialPop* e *trialResults*, respectivamente;
3. Fim
   1. Ao fim de todas as gerações, o elemento com menor valor em *results* é retornado juntamente com o elemento de *pop* que o gerou.

# ANÁLISE DOS RESULTADOS

Com os métodos de Luus-Jaakola e Evolução Diferencial implementados, foram realizadas análises para identificar como os parâmetros de cada método influenciam a estimativa do calor específico da placa de alumínio.

# Luus-Jaakola

A Tabela 1 e a Figura 1 apresentam os resultados obtidos pelo método Luus-Jaakola para diferentes configurações de *nInt*, *nOut* e *coef*.

Tabela – Resultados Luus-Jaakola



A graph of numbers and a bar chart

Description automatically generated with medium confidence

Figura – Resultados Luus-Jaakola

A tabela apresenta os valores de *cp* estimados e os custos associados (*fObj(cp)*) obtidos a partir da variação do coeficiente de contração (*coef*) e do número de iterações internas (*nInt*) e externas (*nOut*) no método Luus-Jaakola. Observa-se que, à medida que *nInt* e *nOut* aumentam, os valores de *cp* convergem gradualmente para um intervalo mais consistente, independentemente do coeficiente de contração utilizado.

Além disso, para *coef*=0.05, os valores de *cp* apresentam maior variação nas iterações iniciais (*nInt*=25, *nOut*=25), enquanto para valores maiores de *coef* (*coef*=0.1 e *coef*=0.2), o *cp* converge de maneira mais estável, especialmente com *nInt* e *nOut* a partir de 75.

Os custos (*fObj(cp)*) permanecem praticamente constantes, com pequenas diferenças decimais que não afetam significativamente a qualidade da solução. Esses resultados indicam que o aumento de *nInt* e *nOut* melhora a precisão na busca pelo valor ideal de *cp*, enquanto o coeficiente de contração exerce menor influência nos estágios avançados da otimização.

# Evolução Diferencial

A Tabela 2 apresenta os resultados obtidos pelo método Evolução Diferencial para diferentes configurações de *nGen*, *F* e *CR* considerando *nPop*=10. Da mesma forma, as Figuras Figura 2, Figura 3, Figura 4 e Figura 5 apresentam esses mesmos resultados para nGen igual a 25, 50, 75 e 100 respectivamente.

Tabela – Resultados Evolução Diferencial



A graph of different colored bars

Description automatically generated

Figura – Resultados Evolução Diferencial (nGen = 25)

A graph of different colored bars

Description automatically generated

Figura – Resultados Evolução Diferencial (nGen = 50)

A graph of numbers and a bar chart

Description automatically generated with medium confidence

Figura – Resultados Evolução Diferencial (nGen = 75)

A graph of different colored bars

Description automatically generated

Figura – Resultados Evolução Diferencial (nGen = 100)

A análise entre as variáveis do método de evolução diferencial (*nPop*, *nGen*, *F* e *CR*) e o valor de *fObj(cp)* indicou que as variáveis *nPop* e *nGen* possuem o maior impacto no valor de *fObj(cp)*. Observou-se que um aumento no tamanho da população e/ou no número de gerações resulta em uma diminuição significativa no valor de fObj(cp). Entretanto na Tabela 2, foi fixado o tamanho da população como 10 para permitir que o impacto das outras variáveis no valor de *fObj(cp)* fosse observado com mais clareza.

Já a taxa de mutação (*F*) apresentou uma influência direta no valor de *fObj(cp)*. Valores mais baixos de *F* geraram valores mais baixos de *fObj(cp)*.

A taxa de cruzamento (CR), por sua vez, não apresentou tanta influência no valor de *fObj(cp)*, mas foi possível observar que valores de *CR* mais próximos a 1 geraram resultados levemente menores.

Portanto, conclui-se que as variáveis mais influentes no valor de *fObj(cp)* são o tamanho da população e o número de gerações, seguidas pela taxa de mutação e pela taxa de cruzamento.

# CONCLUSÃO

Neste trabalho, foram comparados dois métodos de otimização: Evolução Diferencial e Luus-Jaakola, com o objetivo de encontrar o melhor valor para o calor específico da placa de alumínio a fim de que a temperatura calculada a partir desse valor seja mais próxima possível da temperatura coletada no experimento. Observou-se que o método de Evolução Diferencial alcançou o menor valor de *fObj(cp)*, destacando-se pela sua eficácia. No entanto, o método de Luus-Jaakola obteve resultados muito próximos aos da Evolução Diferencial, mas com um consumo significativamente menor de recursos computacionais.

Essa análise demonstra que, embora a Evolução Diferencial tenha se mostrado ligeiramente mais eficaz em termos de valor absoluto da função objetivo, o método de Luus-Jaakola apresentou um melhor custo-benefício devido à sua eficiência computacional.

É importante ressaltar que ambos os métodos requerem uma parametrização adequada para alcançar os melhores resultados possíveis. A correta escolha dos parâmetros é crucial para maximizar a performance e a eficiência dos algoritmos de otimização.

Dessa forma, conclui-se que tanto a Evolução Diferencial quanto o Luus-Jaakola são métodos de otimização viáveis, sendo a escolha do método dependente das especificidades do problema e dos recursos disponíveis.

# Apêndice 1 – Implementação do Luus-Jaakola

A computer screen shot of a program code

Description automatically generated

# Apêndice 2 – Implementação da Evolução Diferencial

A screen shot of a computer program

Description automatically generated

# Apêndice 3 – Implementação da Função Objetivo

A screen shot of a computer program

Description automatically generated

# Apêndice 4 – Implementação do arquivo main.py

A computer screen shot of a program

Description automatically generated

# Apêndice 5 – Dados experimentais

A screen shot of a computer screen

Description automatically generated