COMPARAÇÃO ENTRE SOLUÇÃO DE OTIMIZAÇÃO POR ALGORITMO GENÉTICO E MÉTODO DO FORMIGUEIRO DE UMA FUNÇÃO NÃO-LINEAR

COMPARISON BETWEEN OPTIMIZATION SOLUTION BY GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION OF A NON LINEAR FUNCTION

Lino Timóteo
Discente das Faculdades Integradas de Bauru; Bauru, SP, Brasil; linotim@hotmail.com

Ronaldo César Dametto
Docente e Coordenador do Curso Ciência da Computação das Faculdades Integradas de Bauru, Bauru, SP, Brasil; rdametto@gmail.com

RESUMO
O presente trabalho faz uma comparação de otimização não linear de uma função, também não linear. É usado o Spyder para desenvolvimento de dois algoritmos de otimização, o primeiro é o algoritmo genético e o segundo é o PSO. Algoritmos genéticos tem uma analogia com o processo de seleção natural biológico, assim como os mais aptos sobrevivem e seus genes são herdados a cada geração, o algoritmo genético faz uma "seleção" de valores numéricos, que a cada iteração vai convergindo para a solução ótima. O método do enxame de partículas, PSO, faz uma espécie de "cooperação" entre os pontos afim de que a solução ótima seja encontrada mais rapidamente. Neste segundo método é como um ponto tivesse conhecimento sobre os demais, assim há uma cooperação e não uma competição, como ocorre no caso do algoritmo genético. Existe uma região de um problema com restrições, que é chamada de região factível, onde é possível encontrar a solução ótima. Em solução de problemas de otimização não linear pode existir uma espécie de penalidade, caso essas restrições sejam violadas.
Palavras-chave: Região Factível; Penalidade; Seleção Natural; Cooperação.

ABSTRACT
The present work makes a comparison between non linear optimization of a non linear function too. It uses the Spyder for development of two optimization algorithms, the first is the genetic algorithm and the second is the PSO. Genetic Algorithms have a analogy with the biological natural selection process, likewise the survival of the fittest and their genes are transmitted for each generation, the genetic algorithm makes a "selection" of numerical values, witch for each iteration converges to the optimal solution. The Particle Swarm Optimization method, PSO, makes a kind of collaboration among the points to find the optimal solution faster. In the second method is like if a point knows about the others, therefore exists a collaboration but not a competition, like the genetic algorithm. It exists a region of a problem with constraints which is called feasible region, where is possible to find the optimal solution. In non linear programming can have a certain penalty, in case the constraints be violated.
Keywords: Feasible Region; Penalty; Natural Selection; Collaboration.

1 INTRODUÇÃO

Otimização de funções visa encontrar uma solução ótima para um problema, isto é, uma solução que possa maximizar ou minimizar um problema. A função principal é chamada de função objetivo e possui restrições que não podem ser violadas, pois caso contrário, inviabiliza a solução. Otimizar uma função é importante em várias áreas de pesquisa para se saber o ponto que um problema pode produzir um máximo ou um mínimo, como por exemplo, custo de venda de um produto, custo de uma combinação de geradores elétricos, potência máxima que uma combinação de geradores elétricos pode oferecer, entre outras situações.

Para o estudo de programação linear e não linear é necessário conhecer um pouco sobre a teoria de combinação e conjuntos convexos, que será abordado mais adiante neste artigo. Um estudo sobre programação linear e que serve como base para o estudo de programação não linear pode ser encontrado em [1]. "O conjunto de todas as soluções factíveis do modelo de programação linear é um conjunto convexo" (PUCCINI, 1987, p.70).

A função objetivo é o objeto de estudo a ser analisado. O processo de otimização deve seguir rigorosamente as restrições impostas pelo problema. Caso haja algum desvio das restrições durante o processo de cálculo, existe o método da penalidade que penaliza a função objetivo e esta volta para a região factível para não haver erro na obtenção do ponto ótimo.

Neste trabalho é feito a busca de uma solução ótima, dentro de uma região factível, uma por algoritmo genético e outra pelo método do enxame de partículas. Os dois códigos foram feitos em Python com o ambiente Spyder. Foi usada a biblioteca numpy junto com os recursos *geneticalgorithm* e *pyswarm*, ambos importados.

O algoritmo genético segue uma linha de raciocínio análoga ao estudo da genética, por exemplo o crossing over, que pode ser visto como um processo de mutação, assim esse algoritimo busca uma solução ótima (mais apto no sentido genético). O PSO busca uma solução ótima de um problema, como uma população de pássaros com seus voos aleatórios, mas determinados que levam a um estado evolutivo. Um trabalho envolvendo algoritmo genético com funções não lineares e com o método "Grasshopper Optimization Algorithm" é apresentado em [2]. "[...] called hybrid grasshopper optimization algorithm (GOA) with genetic algorithm (GA): hybrid-GOA-GA [...]". (El-Shorbagy, 2020, p.1). O uso da técnica do "formigueiro" para tratar um problema de otimização não linear é feito em [3]. "Therefore, this proposed algorithm combines PSO (the local search method) and EDA (the global search method) [...]". (WANG, 2020, p.1).

Um estudo mais profundo sobre algoritmos genéticos pode ser encontrado em [4], ao passo que um trabalho sobre enxame de partículas e um pouco de sua teria pode ser encontrado em [5]. O passo a passo dos dois algoritmos pode ser encontrado em [6]. No final deste trabalho é feita a comparação entre os dois métodos para a mesma função objetivo e suas restrições.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 – Conjuntos convexos

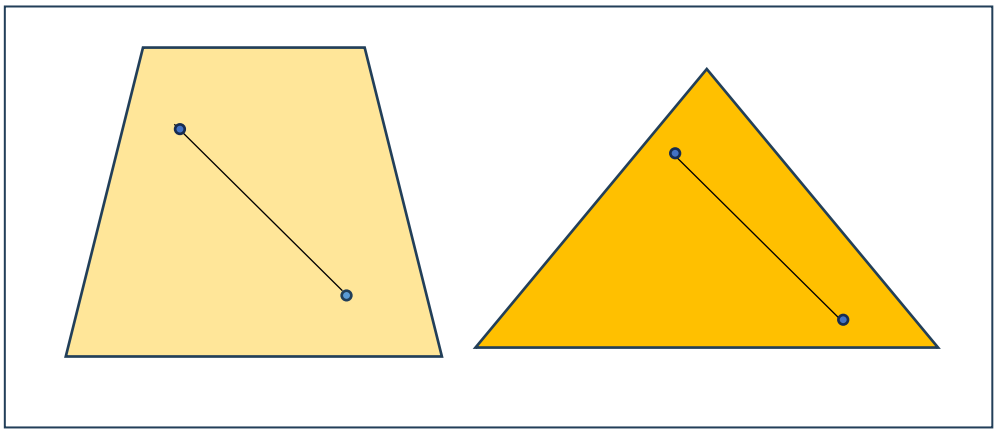
Combinação convexa de vetores é uma combinação linear de vetores, onde os coeficientes somados são numericamente iguais a 1. Esses vetores (aqui no caso são dois vetores), para que formem uma combinação convexa, devem obedecer a uma certa condição. A condição é que todos os pontos que formam a reta que une dois pontos contidos no conjunto convexos, devem também pertencer ao conjunto. O caso para dois vetores é dado a seguir.

Para 2 vetores => $b = \alpha A_1 + (1 - \alpha) A_2$, aqui repare que os coeficientes são α e 1-α, e que somados é numericamente igual a 1. *b* é a combinação convexa e A_1 e A_2 são vetores bidimensionais. O caso pode ser estendido para 3 ou mais vetores, porém a demonstração é mais complexa e no caso de três vetores, fica melhor explicado em [1]. No caso geral para n vetores, a forma geral é a seguinte:

$$\sum_{i=1}^{n} \alpha i = 1 \Rightarrow b = \alpha_1.A_1 + \alpha_2.A2 + ... + \alpha_n.An$$

Conjuntos convexos, como o nome diz é o contrário de côncavo. Esses conjuntos são do tipo pontiagudos, ou seja, não têm cavidade, o que invalidaria a própria definição de convexo. Cada coeficiente α_n deve ser entre zero e um. A figura 1 mostra exemplos de conjuntos convexos para vetores bidimensionais.

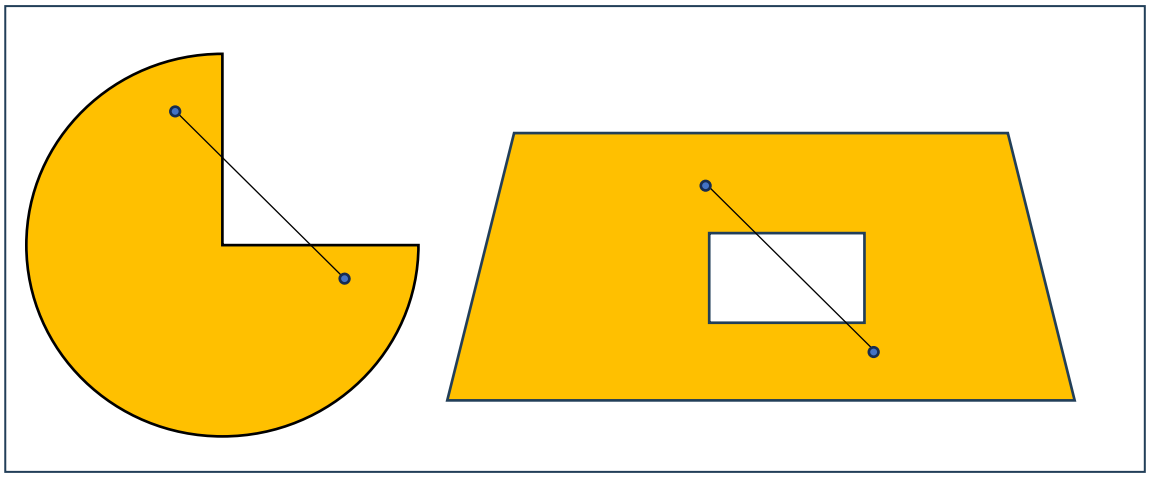
Figura 1 – Exemplos de conjuntos convexos



Fonte: Dados da Pesquisa

Repare na figura 1 que os dois pontos extremos estão dentro do conjunto e além disso, todos os pontos da reta que unem esses dois pontos também pertencem ao conjunto. Isso caracteriza um conjunto convexo. Agora supondo que pontos da reta, que une os dois pontos extremos (vetores), não estejam dentro do conjunto, caracteriza um conjunto não convexo, por ter partes "côncavas", como é o caso da figura 2.

Figura 2 – Exemplos de conjuntos não convexos.



Fonte: Dados da Pesquisa

2.2 – Exemplo literal de um problema de otimização linear

Na figura 3, a região factível é dada pelas condições impostas pelas restrições, elas não podem ser violadas e a solução ótima do problema se encontra nessa região. Um exemplo padrão de como um problema de programação linear pode se apresentar, dado a seguir:

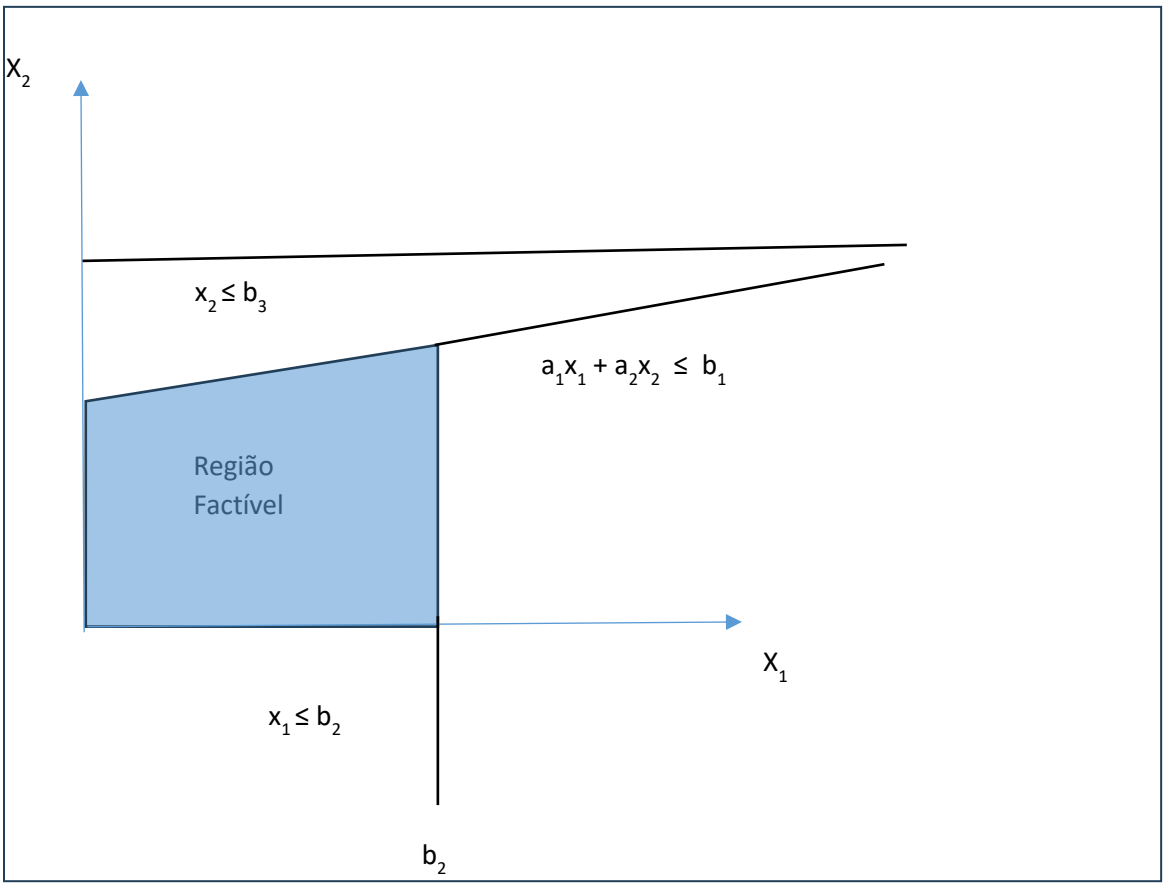
Max. ou Min. : $z = c_1x_1 + c_2x_2$

Sujeito a:

$a_1x_1 + a_2x2 \leq b_1$

$x_1 \leq b_2$

FIGURA 3 – Exemplo de região factível para o problema de otimização linear



Fonte: Dados da Pesquisa

O problema de programação linear pode ser resolvido pelo método simplex. Existem vários métodos para a resolução de problemas de otimização não linear. A forma geral (padrão) para um problema de otimização não linear é dada pela modelo seguinte:

Minimizar f(x)

sujeito a:

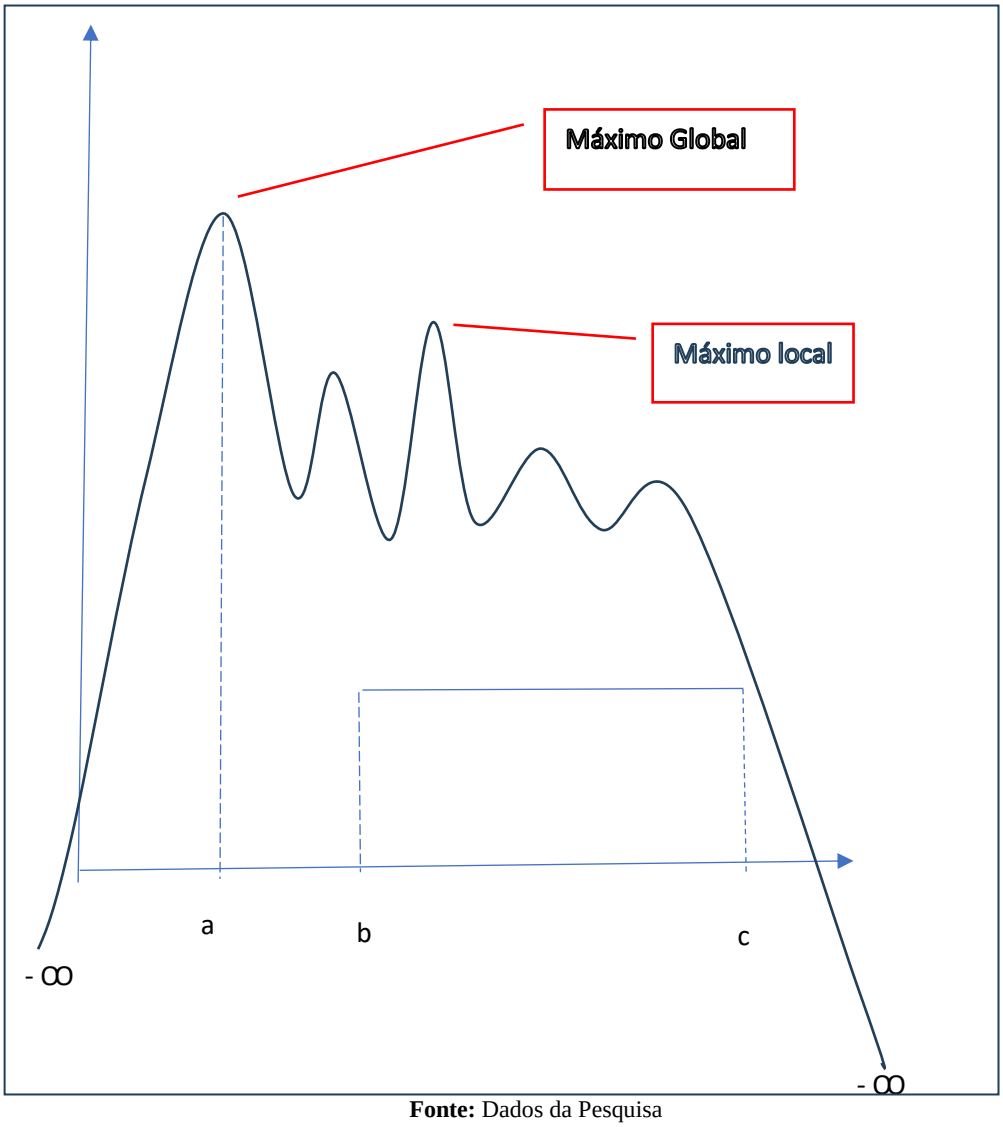
$g_i(x) = 0$

$h_i(x) \leq 0$

2.4 – Método do gradiente

O gradiente é uma derivada que serve para calcular mínimos e máximos de uma função. A busca de uma solução ótima pelo método do gradiente pode apresentar um problema que é ficar retido em um mínimo ou máximo local, segundo [7]. A figura 4 mostra um gráfico de uma função hipotética contendo máximos locais e um global, para um melhor entendimento.

FIGURA 4 – Gráfico de uma função hipotética para máximo local e global



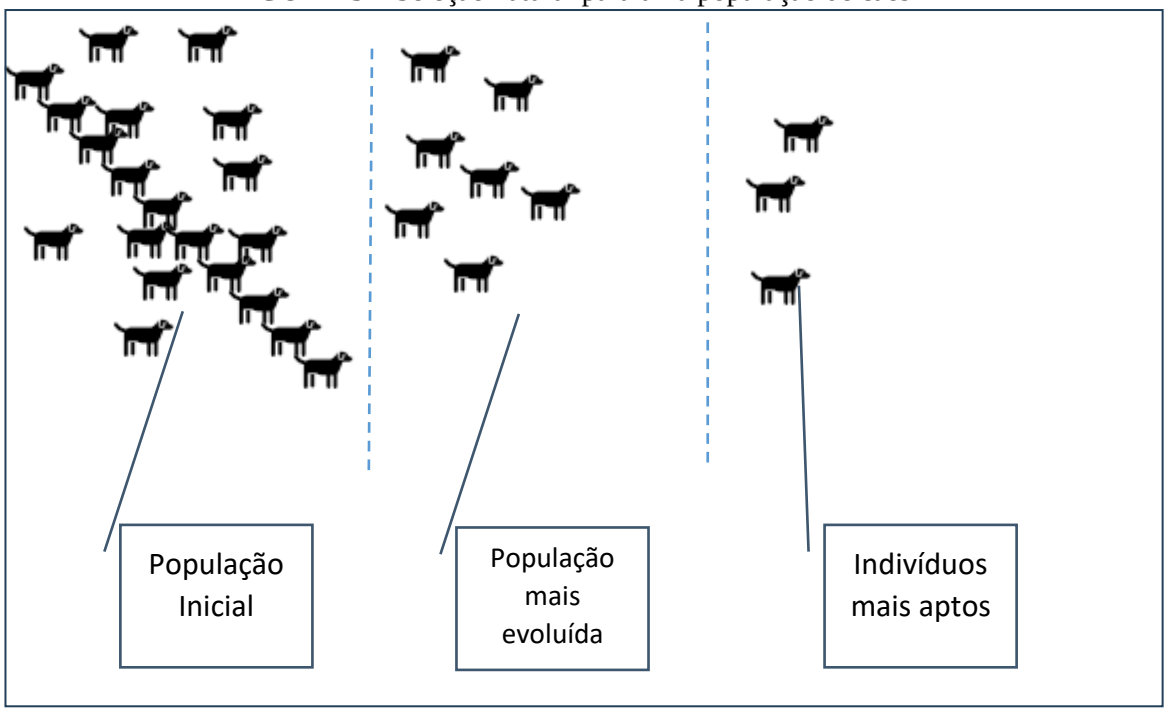
Fonte: Dados da Pesquisa

Na figura 4 existe um máximo local que está compreendido entre os pontos *b* e *c*, esse ponto é máximo relativo, porém existe um máximo global, localizado no ponto *a*, que é o máximo absoluto da função, isto é, o máximo para todo o domínio, desde menos infinito até o mais infinito. O método do gradiente apesenta um problema, como descrito anteriormente, pode ficar retido em um máximo local. E essa é uma vantagem do algoritmo genético, que será visto adiante, e é de otimização global.

2.5 – Algoritmos genéticos

Algoritmos genético seguem uma analogia co a seleção natural do processo evolutivo biológico. Assim como uma população de um ser vivo qualquer pode passar por processos evolutivos, e que são passados para as próximas gerações, através de genes, na computação é como se cada conjunto de pontos a cada iteração ficasse mais próximo da solução ótima. Os pontos vão "evoluindo" até que se tenha a solução que maximiza ou minimiza uma função. Imaginando uma população inicial de cães, assim a cada geração, à medida que alguns indivíduos vão evoluindo, os seus genes são passados para próxima geração, e a cada nova população surgem indivíduos mais aptos, até que apareça uma nova população com as características mais apta, como é ocaso da figura 5.

FIGURA 5 – Seleção natural para uma população de cães

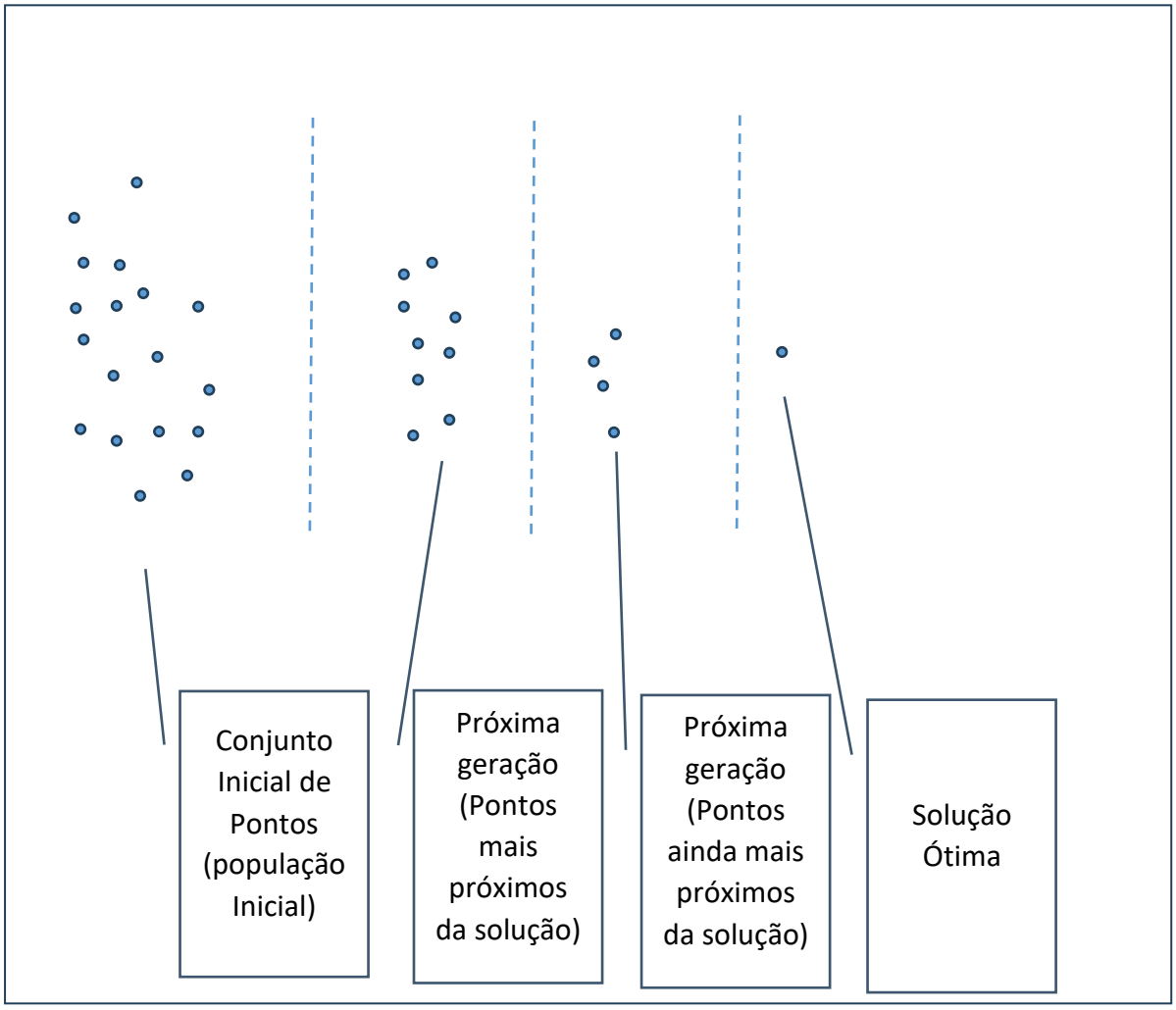


Fonte: Dados da Pesquisa

Analogamente, o algoritmo genético segue um padrão semelhante, com um conjunto de pontos iniciais. O algoritmo faz uma "seleção" durante várias iterações havendo uma busca por pontos que vão se aproximando do ponto ótimo, até que este ponto finalmente seja encontrado. Algo semelhante ao crossover do cromossomo celular ocorre durante a busca pela solução ótima, com uma "taxa de mutação" que é uma característica do algoritmo genético. A figura 6 mostra como isso acontece com um conjunto de pontos. A cada geração vai surgindo pontos

que vão se aproximando do ponto ótimo, sucessivamente, até que seja encontrada a solução ótima.

FIGURA 6 – Conjunto de pontos representando a evolução no algoritmo genético



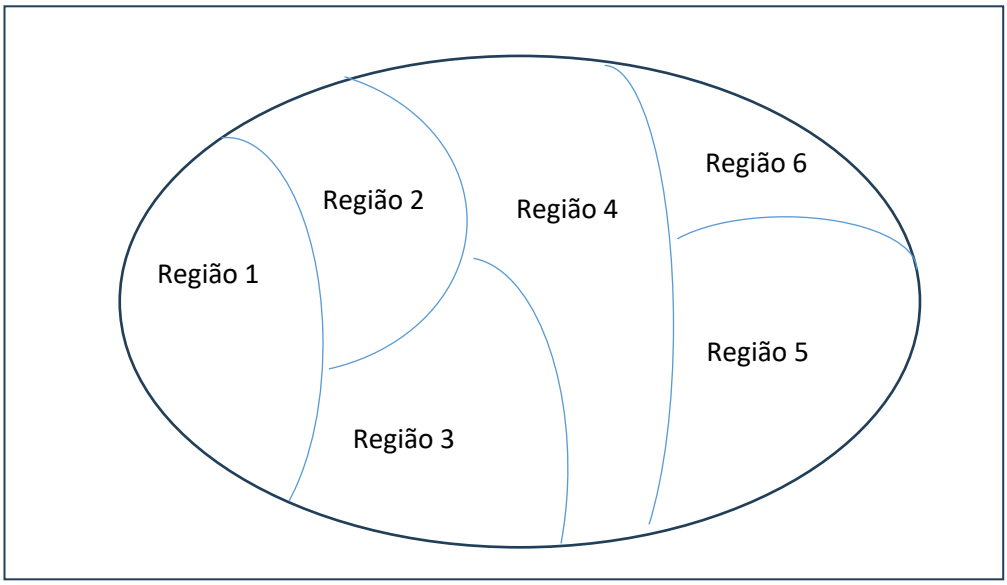
Fonte: Dados da Pesquisa

2.6 – Enxame de partículas (PSO)

O método do enxame de partículas é baseado na cooperação entre indivíduos de uma natureza social. Por exemplo pode ser citado um bando de pássaros com objetivo mutuo, um formigueiro que atua de modo cooperativo, cardume de peixes, enxame de abelhas e qualquer grupo de indivíduos que tenham um objetivo mutuo. Supondo, por exemplo, um barco que afunda em uma determinada área do mar. Supondo ainda que um grupo de mergulhadores estejam dispostos a procurar pelo barco, afim de encontra-lo. Então a área em que o barco possa ser encontrado poderá ser dividida em regiões ou subáreas e cada grupo procura em uma subárea. Se um grupo não achou, os outros grupos sabem que naquela região não está e procurarão em outras áreas. Assim o trabalho é dividido e feito de modo mais rápido acelerando

a busca. No método de enxame de partículas é como se cada ponto tivesse conhecimento dos outros pontos e então há uma analogia com o processo de cooperação. A figura 7 mostra uma área sendo dividida em subáreas, então cada grupo fica responsável por uma região, acelerando a procura de modo cooperativo.

FIGURA 7 – Área do oceano para a busca de um hipotético barco afundado



Fonte: Dados da Pesquisa

3 MATERIAIS E MÉTODOS

Para a simulação dos dois métodos para a função não linear, foi usado a biblioteca numpy da linguagem de programação Python pelo ambiente Spyder. Primeiramente foi feita a simulação do algoritmo genético importando o *geneticalgorithm* e posteriormente importando o *pyswarm* para o método do enxame de partículas. A função objetivo e suas restrições é apresentada a seguir:

$$z = x^2 + xy$$

s.a:

$-x + 3xy \leq 10$

$2x - 4y \leq 15$

$xy + 3y \leq 12$

$-x \leq 0$

$-y \leq 0$

$x \leq 6$

$y \leq 7$

Nos dois códigos que são apresentados a seguir, usam uma penalidade. O método da penalidade transforma um problema com restrições em um problema sem restrições equivalente. Equivalente aqui subtende-se como problemas que têm a mesma solução. Durante o processo iterativo, caso a função objetivo seja violada, isto é, desvio da região factível ou restrições, a função é penalizada.

3.1 – Código para o método do algoritmo genético

```
import numpy as np
from geneticalgorithm import geneticalgorithm as ga
def fobj(x):
    pen = 0
    if not -x[0] + 3*x[0]*x[1] <= 10: pen = np.inf
    if not 2*x[0] - 4*x[1] <= 15 :  pen = np.inf
    if not x[0]*x[1] + 3*x[1] <= 12 : pen = np.inf
    return -(x[0]**2 + x[0]*x[1]) + pen
varbounds = np.array([[0,6], [0,7]])
vartype = np.array([['int'], ['real']])
algorithm_param = {'max_num_iteration':100, \
                   'population_size':1000, \
                   'mutation_probability': 0.1,\
                   'elit_ratio': 0.01,\
                   'crossover_probability': 0.5,\
                   'parents_portion': 0.3, \
                   'crossover_type':'uniform',\
                   'max_iteration_without_improv':None}
modelo = ga(function=fobj, dimension=2,
            variable_type_mixed=vartype,
            variable_boundaries=varbounds,
            algorithm_parameters=algorithm_param)
modelo.run()
```

Um trecho importante deste código é o dicionário que ele contém. Nesse dicionário é possível manipular as grandezas que compõem o algoritmo genético. Este dicionário pode ser encontrado em um link exposto em [6]. Esse dicionário é apresentado a seguir:

```
algorithm_param = {'max_num_iteration':100, \
                   'population_size':1000, \
                   'mutation_probability': 0.1,\
                   'elit_ratio': 0.01,\
                   'crossover_probability': 0.5,\
                   'parents_portion': 0.3, \
                   'crossover_type':'uniform',\
                   'max_iteration_without_improv':None}
```

3.2 – Código para o método de enxame de partículas

```
import numpy as np
from pyswarm import pso
def fobj(x):
    pen =0
    x[0] = np.round(x[0],0)
    if not -x[0] + 3*x[1]*x[0]  <= 10: pen = np.inf
    if not 2*x[0] - 4*x[1] <= 15: pen = np.inf
    if not x[0]*x[1] + 3*x[1] <= 12: pen = np.inf
    return -(x[0]**2 + x[1]*x[0]) + pen
def const(x):
    return []
lb = [0,0]
ub = [6,7]
x0 = [0,0]
xopt, fopt = pso(fobj, lb, ub, x0, const)
print('x = ', xopt[0])
print('y = ', xopt[1])
print('A funao objetivo vale:', fobj(xopt))
```

O código para o método do enxame de partículas é mais simples, não tem tantos parâmetros como o método por algoritmo genético. Neste código também é usada uma função penalidade, como pode ser vista no código, caso a função objetivo desvie da região factível. Um detalhe maior sobre os dois métodos pode ser encontrado em [6].

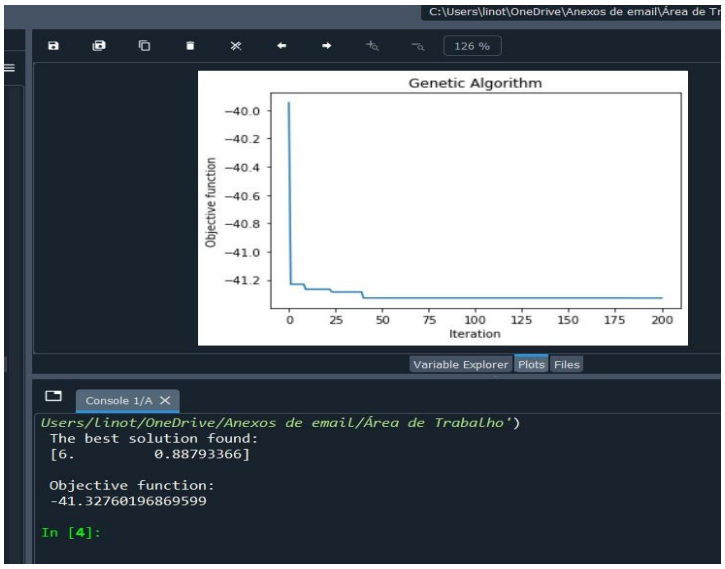
4 RESULTADOS E DISCUSSÃO

Para validar os dois métodos, foram feitas duas simulações para comparar os resultados obtidos. Para o método do algoritmo genético foram feitas duas simulações, a primeira com 200 iterações e a segunda com 1000 iterações. Depois foi feita uma simulação pelo método do enxame de partículas para ver o comportamento do resultado. o sinal negativo da função objetivo significa que ela foi minimizada. Minimizar a função objetivo é o mesmo que a maximizar com o sinal contrário.

4.1 – Resultados da simulação por algoritmo genético

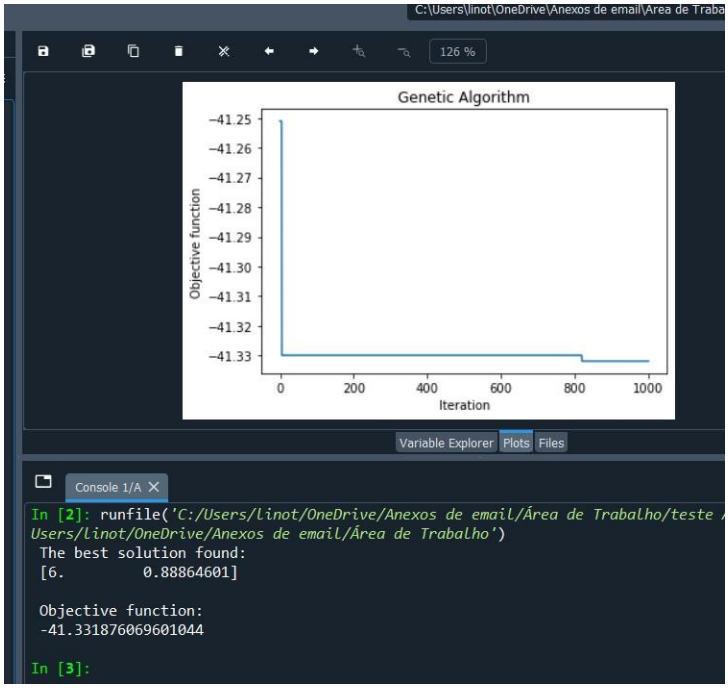
Os resultados foram bem próximos para 200 e 1000 iterações. O algoritmo genético pode ter resultados não exatamente iguais, mas com muita pouca diferença, pois os parâmetros podem ser mudados, e é um processo aleatório. Para 200 iterações, o resultado convergiu a partir da iteração número 50. Para 1000 iterações houve uma convergência a partir da iteração número 800.

FIGURA 8 – Valor ótimo para 200 iterações por algoritmo genético



Fonte: Dados da Pesquisa

FIGURA 9 – Valor ótimo para 1000 iterações por algoritmo genético



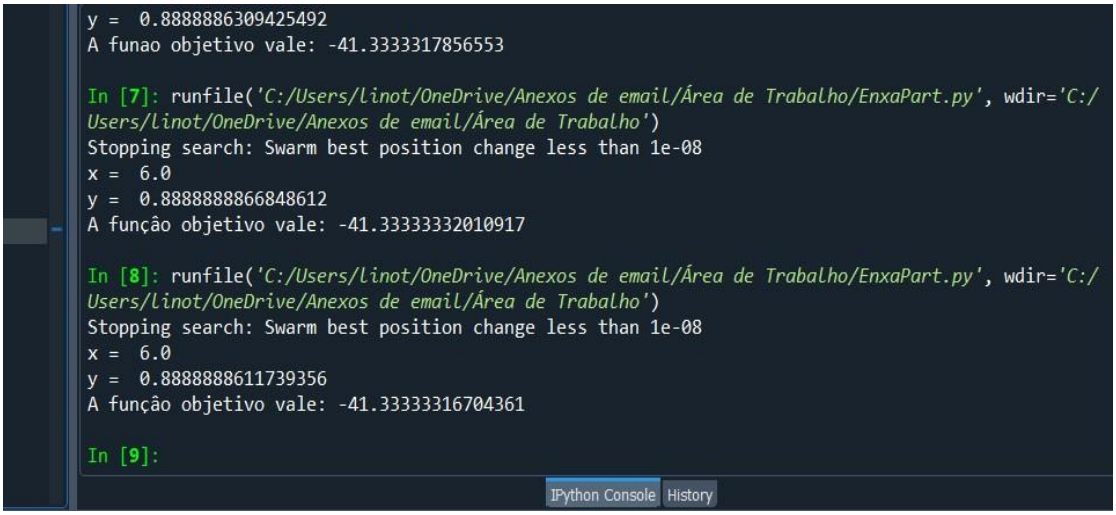
Fonte: Dados da Pesquisa

4.2 – Resultados da simulação por enxame de partículas

O enxame de partículas parte de uma população (pontos) aleatória. Sendo assim, em diferentes simulações, como mesmo código, é possível encontrar valores diferenteS para a função objetivo. Esses valores podem divergir a partir da sétima, oitava ou outra casa decimal,

mas a precisão é boa. É o caso da figura 10, em que o código foi simulado duas vezes e a função objetivo apresentou divergência a partir da sétima casa decimal.

FIGURA 10 – Valor ótimo da função objetivo para duas simulações com o mesmo código



Fonte: Dados da Pesquisa

5 CONCLUSÕES

1 – O método do enxame de partículas convergiu mais rápido que o método do algoritmo genético.

2 – A diferença de tempo de execução pode estar associada ao fato que o algoritmo genético é baseado em seleção (competição entre os pontos), enquanto o enxame de partículas usa um processo de cooperação.

3 - Os dois métodos resultaram na solução ótima com valores bem próximos, validando os dois modelos.

REFERÊNCIAS

[1] PUCCINI, Abelardo de Lima; PIZZOLATO, Nélio Domingues. **Programação linear**. Rio de Janeiro, São Paulo: Livros Técnicos e Científicos Editora SA, 1987.

[2] EL-SHORBAGY, M. A.; EL-REFAEY, Adel M. **Hybridization of Grasshopper Optimization Algorithm With Genetic Algorithm for Solving System of Non-Linear Equations**. IEEE Accsess, [*S. l.*], p. 1-18, 07 dez. 2020. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9285252. Acesso em: 20 jun. 2023.

[3] WANG, Guangmin; MA, Linmao. **The Estimation of Particle Swarm Distribution Algorithm With Sensitivity Analysis for Solving Nonlinear Bilevel Programming Problems.** IEEE Accsess, [*S. l.*], p. 1-17, 21 jun. 2020. Disponível em:

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9145519. Acesso em: 20 jun. 2023.

[4] LEOW, Peter. **Genetic Algorithms Demystified**: Unravel the Myths and Power of Genetic Algorithms in Machine Learning. [*S. l.*: *s. n.*], 2016. *E-book*.

[5] OLIVEIRA, Marilyn Cristine Serafim de; SILVA, Thales Lima; ALOISE, Dario José. **Otimização por nuvem de partículas: diferença entre aplicações a problemas contínuos e discretos**. XXXVI-SBPO, [*s. l.*], 2004. Disponível em: http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/OtimizacaoPorNuvemParticulas.pdf. Acesso em: 20 jun. 2023.

[6] **Otimização em Python:** Pesquisa Operacional. [*S. l.*], [2020]. Disponível em: https://www.udemy.com/course/otimizacao-em-python/learn/lecture/20998818?start=435#overview. Acesso em: 20 jun. 2023.

[7] **Redes Neurais Artificiais em Python. Direção**: Jones Granatyr. [*S. l.*: *s. n.*], 2022. Disponível em: https://www.udemy.com/course/redes-neurais-artificiais-em-python/learn/lecture/8048696#overview. Acesso em: 16 jun. 2023.