# Relatório do Laboratório 5: Estratégias Evolutivas

Isabelle Ferreira de Oliveira
*CT-213 - Engenharia da Computação 2020*
*Instituto Tecnológico de Aeronáutica (ITA)*
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

***Resumo*—Esse relatório documenta a implementação de uma estratégia evolutiva simples e comparação de seu desempenho com o de CMA-ES em funções usadas como *benchmark* para algoritmos de otimização. Essas funções utilizadas eram uma esfera transladada, e as fuções de Ackley, Schaffer 2 e Rastrigin 2D.**

***Index Terms*—Estratégias evolutivas, CMA-ES, *benchmark*, Ackley, Schaffer 2, Rastrigin 2D**

## I. Introdução

Otimização consiste em encontrar o mínimo (ou máximo) de uma função, ou seja, encontrar o conjunto de parâmetros que levem essa função ao seu mínimo (ou máximo). Também pode ser visto como encontrar a melhor solução dentre todas as soluções viáveis. Nesses problemas de otimização também é possível haver restrições acerca dos parâmetros que serão analisados.

Dentre os mais diversos tipos de algoritmos de otimização, existem os métodos baseado em população. Esses métodos mantêm uma população de possíveis soluções, a fim de tentar encontrar diferentes mínimos locais da função a ser analisada, melhorando assim as chances de se obter uma solução promissora. Um exemplo famoso de algoritmo de otimização baseado em população é o *Particle Swarm Optimization* (PSO).

O pseudo-código desse algoritmo para maximização pode ser visto na subseção a seguir. Em seguida, será apresentado como esse algoritmo foi implementado no contexto do laboratório.

### A. Particle Swarm Optimization (PSO)

Será considerado uma classe *ParticleSwarmOptimization*, representando o algoritmo PSO. Além disso, o pseudocódigo do uso do algoritmo a partir dessa classe é o mostrado a seguir. Nesse pseudocódigo, *pso* é o objeto da classe *ParticleSwarmOptimization*.

```
for i in range(num_evaluations):
  position = pso.get_position_to_evaluate()
  value = quality_function(position)
  pso.notify_evaluation(value)
```

No pseudocódigo acima, *quality_function()* é a função a ser maximizada. Assim, iterativamente por tantas vezes até se atingir uma convergência satisfatória ao(à) programador(a), o algoritmo calculará o valor da função para cada partícula na população de candidatas à solução. Para cada partícula, então, após esse cálculo, a função *notify_evaluation()* compara o resultado obtido aos obtidos anteriormente a fim de encontrar o valor maximizado. Uma ideia de implementação em pseudocódigo da função *notify_evaluation()* foi apresentada a seguir.

```
def notify_evaluation(value):
  current_particle =
      get_current_particle_to_evaluate()

  if value > current_particle.my_best_value:
    current_particle.my_best_value = value
    current_particle.my_best_position =
        current_particle.position

  if value > global_best_value:
    global_best_value = value
    global_best_position =
        current_particle.position

  num_particle_evaluated =
      num_particle_evaluated + 1
  if num_particle_evaluated == num_particles:
    num_particle_evaluated = 0
    advance_generation()
```

A função *advance_generation()* é a responsável por atualizar os valores de cada partícula a cada geração de população. As posições de cada particula são acrescidas de uma velocidade para gerar as posições a serem analisadas na geração seguinte, e o intuito dessa velocidade é conduzir as partículas a posições cada vez mais próximas da solução otimizada.

Essa velocidade foi calculada a partir da equação fornecida a seguir, na qual $\omega$ é relativo a inércia dessa mudança, $\varphi_p$ e $\varphi_g$ são, respectivamente, os parâmetros cognitivo e social, e $b$ e $b_g$ são, respectivamente, as melhores posições a nível dessa partícula em específica e a nível global (entre todas as partículas). Além disso, existem os parâmetros $r_p$ e $r_g$, que conferem aleatoriedade ao sistema.

$$v = \omega \cdot v + \varphi_p \cdot r_p \cdot (b - x) + \varphi_g \cdot r_g \cdot (b_g - x)$$

Abaixo, por fim, está apresentado o pseudocódigo para a ideia acima.

```
def advance_generation():
  for particle in particles:
    r_p = random.uniform(0, 1)
```

```
r_g = random.uniform(0, 1)

particle.velocity = inertia_weight *
    particle.velocity +
    cognitive_parameter * r_p *
    (particle.my_best_position -
    particle.position) + social_parameter
    * r_g * (global_best_position -
    particle.position)

for i in range(quantity_of_dimensions):
  delta = upper_bound[i] - lower_bound[i]
  particle.velocity[i] =
      min(max(particle.velocity[i],
      -delta), delta)

particle.position = particle.position +
    particle.velocity

for i in range(quantity_of_dimensions):
  particle.position[i] =
      min(max(particle.position[i],
      lower_bound[i]), upper_bound[i])
```

## II. Implementação do algoritmo

Na parte relativa a implementação do algoritmo PSO, era necessário preencher os construtores das classes *Particle* e *ParticleSwarmOptimization*, além dos códigos das funções *get_best_position()*, *get_best_value()*, *get_position_to_evaluate()*, *advance_generation()* e *notify_evaluation()* da classe *ParticleSwarmOptimization*. Esses códigos a se completar estavam todos no código base fornecido [1]. Além disso, era necessário completar também o código da função *evaluate()* da classe *Simulation*, função essa a ser otimizada para o problema do robô seguidor de linha.

A análise de vários pontos dos algoritmos descritos acima terão uma breve descrição em alto nível da sua implementação a seguir.

Primeiramente, foi criada uma lista de particulas para a classe principal *ParticleSwarmOptimization*, um contador referente a qual partícula dessa lista o código estará se referindo em uma determinada iteração, além de variáveis para guardar o melhor valor (referente a função qualidade a ser otimizada) encontrado até aquela iteração dentre todas as partículas e qual a posição desse melhor valor.

Essa lista de partículas é formada por objetos da classe *Particle*, e cada partícula contém sua posição, sua velocidade, além da melhor posição referente ao valor calculado da função de qualidade e qual esse melhor valor alcançado pela partícula. A posição e a velocidade inicial de cada partícula é escolhida uniformemente aleatória, respeitando os limites estabelecidos por quem chama o construtor dessa classe.

As funções *get_best_position()* e *get_best_value()* retornam, respectivamente, a posição referente ao melhor valor encontrado na função qualidade e qual esse melhor valor dentre todas as partidas e todas as iterações. Já a função *get_position_to_evaluate()* se utiliza do contador citado anteriormente para retornar o elemento correto na lista de partículas, ou seja, a próxima partícula a ser analisada.

Por fim, as funções *advance_generation()* e *notify_evaluation()* foram implementadas conforme apresentado nos pseudo códigos da seção Introdução, com alguns detalhes como utilizar o contador referente a qual partícula da lista de partículas o código está se referindo na iteração em questão em *notify_evaluation()*. Além disso, para o caso do robô seguidor de linha, a função *evaluate()* da classe *Simulation* foi implementada conforme o apresentado na Seção 2 do roteiro do laboratório [1], com $e_k = 1$ para o caso de não detecção de linha (booleana *detection* sendo *False*) e $\omega = 0.5$.

## III. Resultados e Conclusões

A otimização foi executada para duas funções de qualidade: uma função matemática $f(x) = -(x(0)-1)^2 + (x(1)-2)^2 + (x(2)-3)^2$ e a função *evaluate()* do robô seguidor de linha. Os resultados das otimizações obtidas após a execução da implementação do algoritmo descrito acima foram apresentados nas Figuras de 1 a 4 para a função matemática, e de 5 a 8 para o robô seguidor de linha.

### A. Esfera Transladada

A partir da Figura 2, foi possível notar o correto funcionamento da implementação do algoritmo PSO, uma vez que os parâmetros $x(0)$, $x(1)$ e $x(2)$ convergiram corretamente para os valores 1, 2 e 3, como era esperado. Além disso, as Figuras 3 e 4 comprovaram a convergência da função para 0, valor máximo também já esperado.

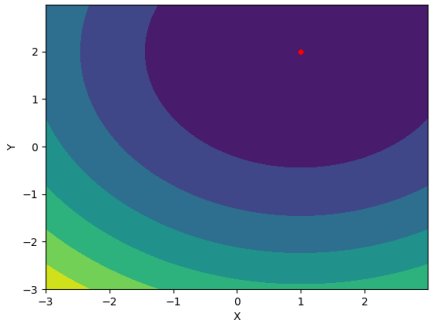


Figura 1. Valores convergidos dos parâmetros para maximização da função para o caso da função matemática.

### B. Robô seguidor de linha

Tendo em vista o correto funcionamento do algoritmo PSO dado o apresentado na subseção anterior e o comportamento coerente realizado pelo robô apresentado na Figura 8, pode-se concluir que os resultados apresentados na Figura 5 são parâmetros satisfatórios do problema, chegando a convergência maximizada da Figura 7. Esses valores de parâmetros foram encontrados após cerca de 3000 iterações, levando a um valor de função qualidade máximo de 572.33; parâmetros esses: a

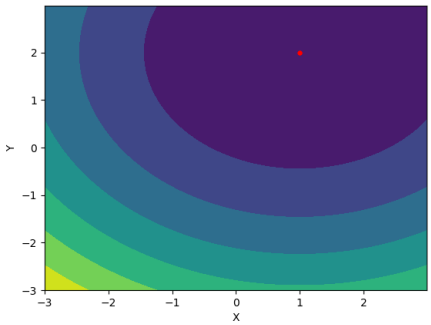


Figura 2. Valores convergidos dos parâmetros para maximização da função para o caso da função matemática.

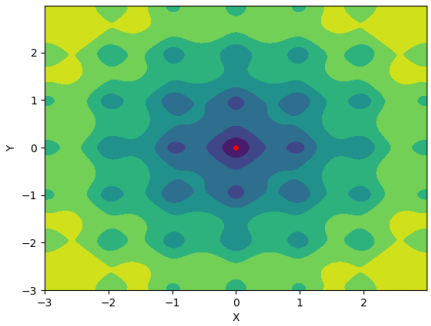


Figura 3. Convergência do valor da função qualidade com o passar das iterações para o caso da função matemática.

velocidade linear do robô (0.72) e os ganhos proporcional, integrativo e derivativo (131.73, 624.82, 15.37, respectivamente).

Tendo em vista o que foi apresentado, pode-se notar, por fim, que esse algoritmo realmente se demonstrou eficaz em encontrar parâmetros otimizados para uma determinada função de custo e um conjunto de possíveis soluções iniciais.

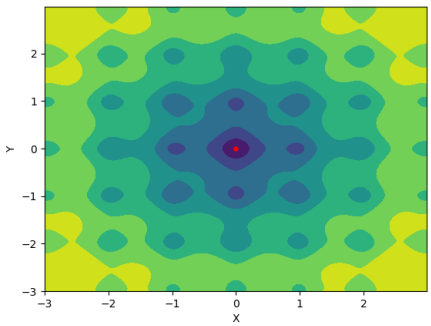


Figura 4. Convergência do melhor valor da função qualidade encontrado com o passar das iterações para o caso da função matemática.

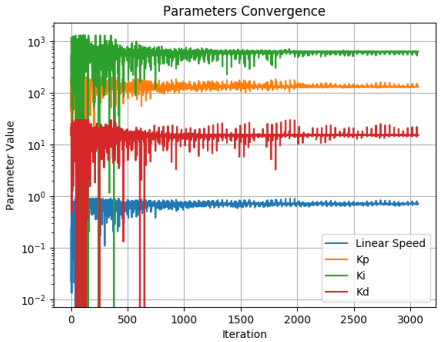


Figura 5. Valores convergidos dos parâmetros para maximização da função para o caso do robô seguidor de linha.

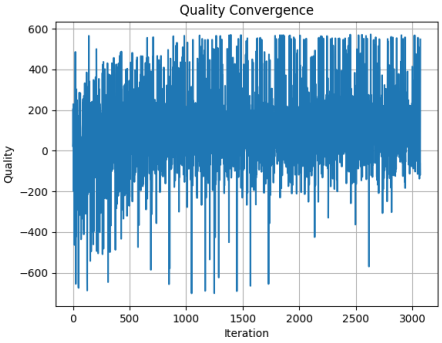


Figura 6. Convergência do valor da função qualidade com o passar das iterações para o caso do robô seguidor de linha.

## Referências

[1] M. Maximo, "Roteiro: Laboratório 4 - Otimização com Métodos Baseados em População". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

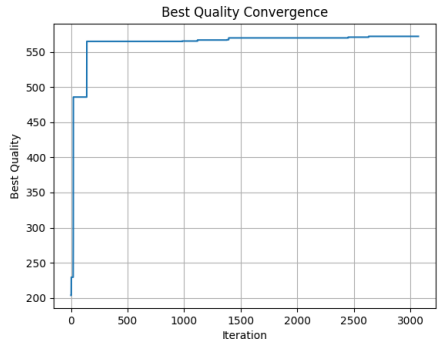


Figura 7. Convergência do melhor valor da função qualidade encontrado com o passar das iterações para o caso do robô seguidor de linha.

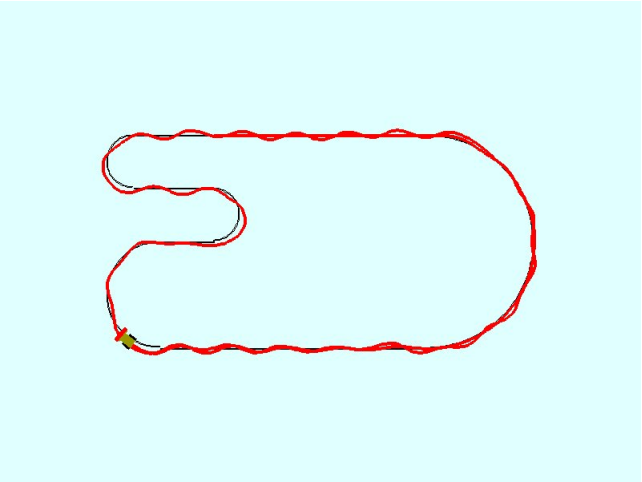

Figura 8. Simulação do robô seguidor de linha realizando o percurso com os valores de parâmetros encontrados a partir da otimização por PSO.