1.Comparação entre o Algoritmo Genético implementado usando uma estrutura de dados binária e real

Conforme apresentado no trabalho 3, um algoritmo genético é composto de algumas fases, que são:

1 - Geração da população inicial
2 - Avaliação
3 - Até a condição de parada ser alcançada
    3.1 - Gera uma nova população
        3.1.1 - Seleção
        3.1.2 - Crossover
        3.1.3 - Mutação
    3.2 - Avaliação

Os passos são os mesmos com uma estrutura binária e real, o que muda é a implementação desses passos.

A maioria das mudanças trouxe uma simplificação ao código de muitos métodos e remoção de alguns como podemos ver nos exemplos abaixo.

```
def _gerarIndividuos(self):
    from random import randint
    return [
        Individuo(
            [randint(0,1) for _ in range(self.precisao)]
        ) for _ in range(self.qtdIndividuos)
    ]
    from random import uniform
    return [Individuo(uniform(self.dominio[0], self.dominio[1])) for _ in range(self.qtdIndividuos)]
```

Acima é mostrado o bloco removido (vermelho) e o adicionado(verde). Ao invés de usar duas estruturas de repetição para criação de um indivíduo passei a utilizar somente uma.

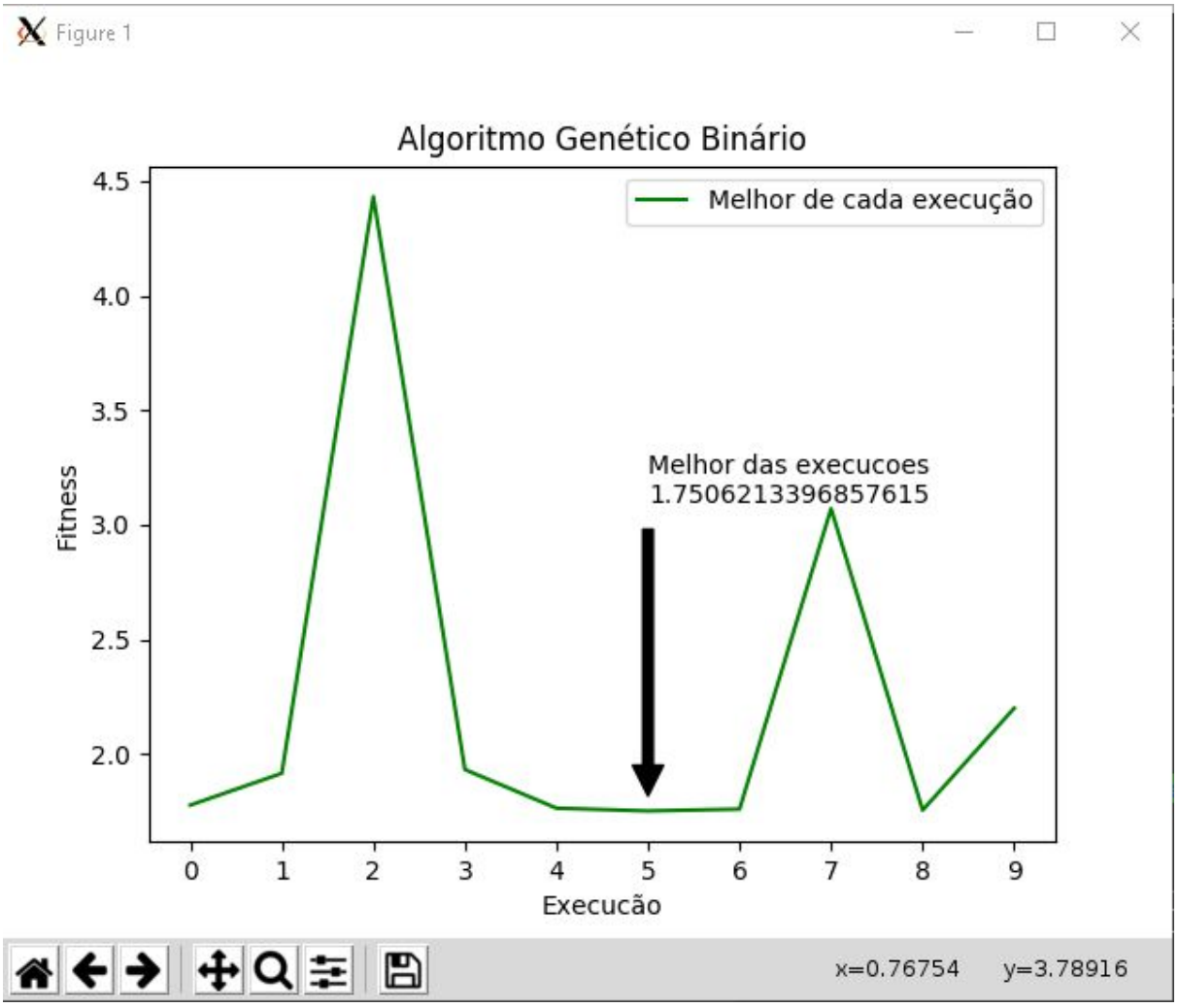
```python
class Individuo:
    def __init__(self, cromossomo):
        self.cromossomo = cromossomo
        self.fitness = None

    def __eq__(self, value):
        return self.fitness == value.fitness

    def __le__(self, value):
        return self.fitness <= value.fitness

    def __lt__(self, value):
        return self.fitness < value.fitness

    def cromossomo_str(self):
        return ''.join(map(str, self.cromossomo))

    def cromossomo_int(self):
        return int(self.cromossomo_str(), 2)
```

Da classe Indivíduo forem removidos os métodos que retornam o valor do cromossomo como um inteiro ou string, já que o mesmo era armazenado em forma binária. O método de normalização também foi removido já que o valor é armazenado sem que seu valor precise ser alterado.

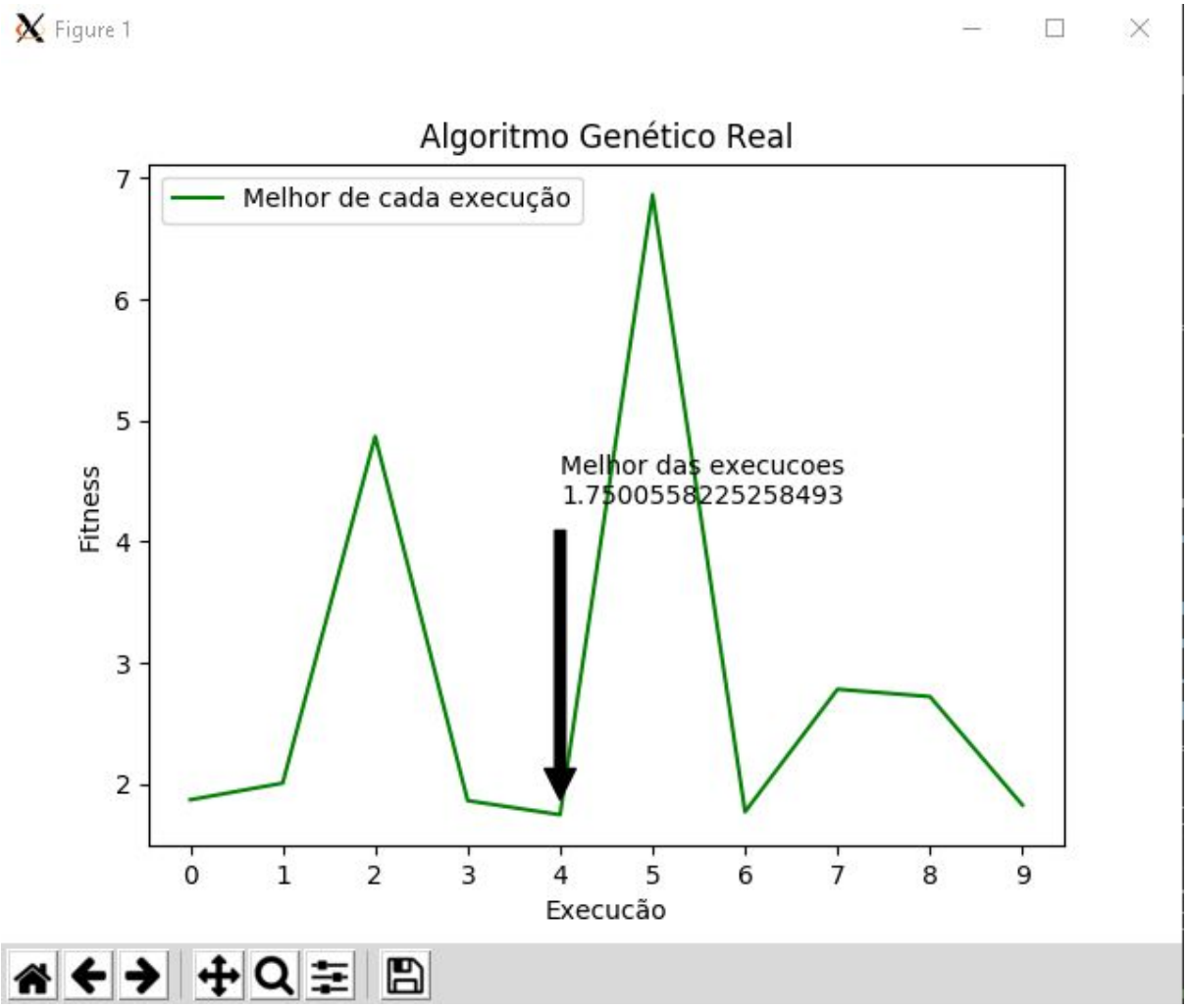
Sobre os métodos principais do algoritmo, o de seleção permaneceu inalterado pois continuou atuando da mesma forma, pois as propriedades da classe Indivíduo e como eram comparados também não mudou. Na parte de crossover foi usado o Blend Crossover (BLX-α), porém não sei se foi feita corretamente para gerar dois filhos com um único parâmetro por indivíduo. A mutação ficou por conta da Mutação de Limite, que é bem simples de ser implementada.

2. Resultados

Abaixo podemos ver o resultado da execução do algoritmo em sua versão binária e real. Lembrando que em nenhuma das versões foi feita com elitismo.



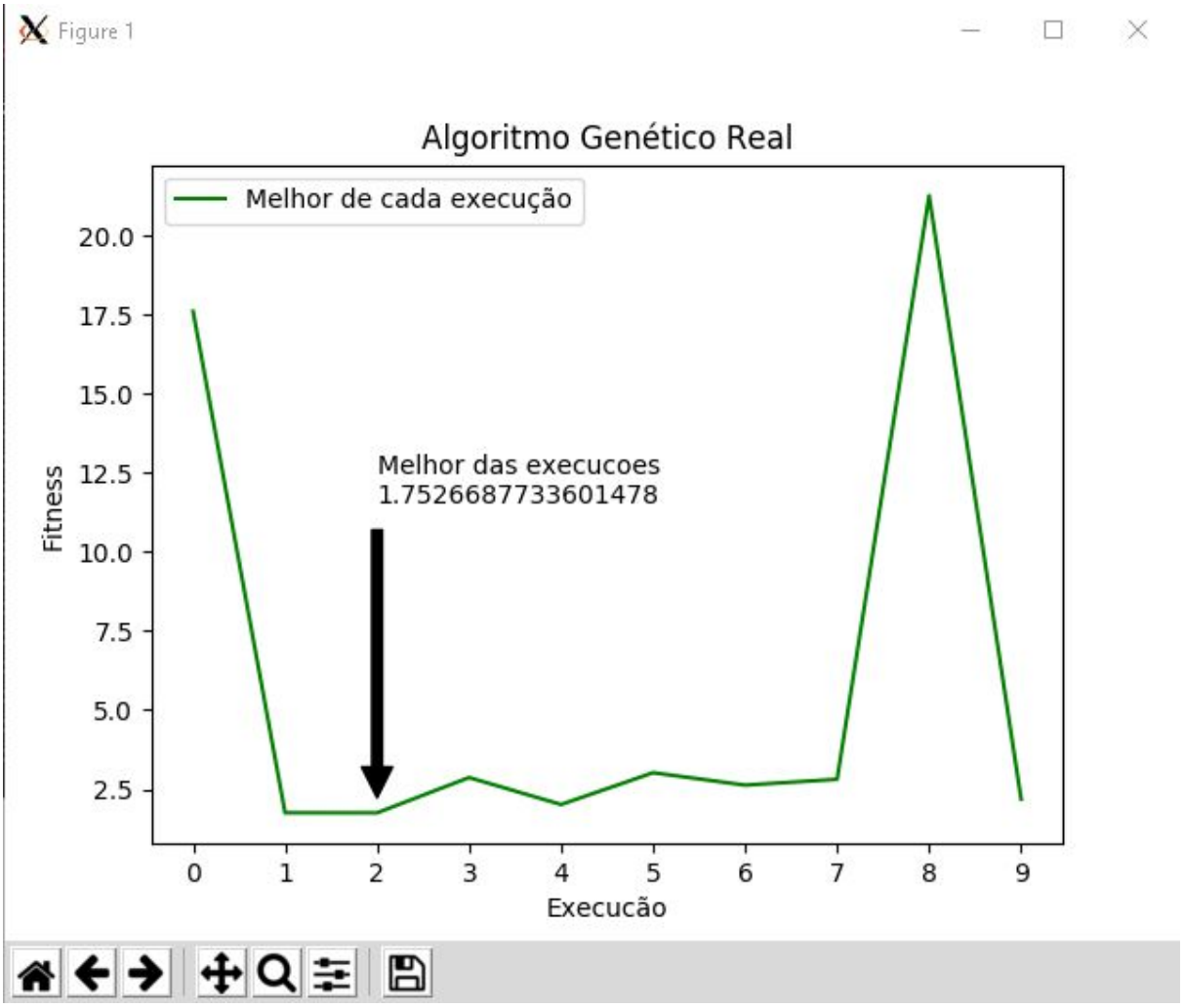
Algoritmo Genético Binário



Algoritmo Genético Real

Pode ser visto que a versão real levou a melhor no resultado da melhor de suas execuções. Cada execução mostrada no gráfico representa exatamente isso, uma execução completa do algoritmo, com a melhor média das gerações. Por isso a variação nos valores de Y, não é possível controlar o valor das execuções.

No Blend Crossover BLX-α é possível escolher se quer que o valor de Beta seja mantido para todos filhos ou que seja gerado um novo. Fiz um execução com esse valor falso a fim de perceber alguma mudança, mas não a identifiquei. Talvez seja pela forma como os resultados são apresentados, será testado plotando apenas os valores de cada geração, para acompanhar como ocorre o distanciamento dos filhos. Será testado também a mutação gaussiana em futuras alterações do código.



Algoritmo Genético Real - Valor de Beta gerada para cada Filho

Por fim é possível concluir que a representação real tem vários benefícios em relação a binária, como uma complexidade menor na montagem dos algoritmos, menos funções auxiliares e mais opções de crossover e mutação.