Trabalho Prático 1 - Programação Genética

Marcelo Sartori Locatelli

1 Introdução

Programação genética é uma técnica de design de algoritmos muito útil para encontrar soluções para diversos problemas, sobretudo do mundo real. Em geral ela consiste na criação de uma população inicial de indivíduos (representações de soluções) aleatórios, que, ao longo de várias iterações, vai se modificando de acordo com operadores genéticos, guiada por uma medida de aptidão, comumente chamada de *fitness*, essas modificações funcionam como uma forma de exploração do espaço de busca do problema original. É um método não determinístico e, portanto, execuções diferentes podem levar a resultados diferentes.

Uma de suas principais aplicações é a chamada regressão simbólica, que consiste em encontrar uma expressão simbólica que melhor aproxima uma função desconhecida a partir de um conjunto de amostras. Nesse trabalho, será discutida a implementação e subsequente experimentação da regressão simbólica a partir do uso de um algoritmo de programação genética. No contexto de um algoritmo de programação genética, a experimentação é essencial, visto que seu funcionamento depende de uma série de parâmetros - tamanho da população, número de gerações, probabilidade de operadores genéticos, etc - que devem ser determinados experimentalmente.

2 Linguagem Utilizada

Para a implementação desse algoritmo foi usada a linguagem Python 3.9 com a biblioteca externa Numpy. As analises foram feitas usando Jupyter Notebook e o arquivo Main.py com as bibliotecas numpy, pandas e matplotlib.

Para a execução do algoritmo em sua forma mais simples(fora dos notebooks), pode-se usar:

python Main.py PopSize Generations Crossover Mutation TournamentSize Elitism Path_To_Train Path_To_Test seed(optional)

onde popsize, generations, tournamentsize e seed são inteiros, Crossover e Mutation são floats tais que Crossover+Mutation ≤ 1 e elitism é True ou False.

Os notebooks requerem que as bases de dados estejam na mesma pasta do código.

3 Implementação

Para a implementação de um algoritmo de programação genética são necessárias diversas decisões de projeto, tais como a representação(genótipo) de um individuo, quais terminais e não terminais serão usados.

A figura 1[1] representa um fluxograma de um algoritmo genérico de programação genética. Nele, pode-se ver as diversas etapas de um algoritmo dessa natureza.

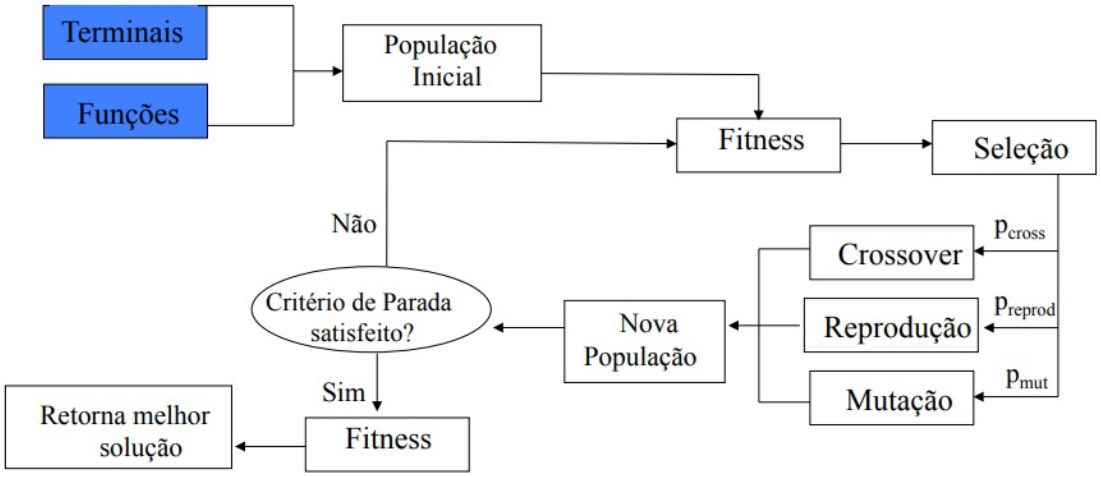


Figura 1: Fluxograma da programação genética

3.1 Símbolos

Antes de gerar qualquer árvore de expressões, é necessário definir quais símbolos farão parte dela. Evidentemente, os terminais deverão ser constantes e as variáveis da função, porém, os operadores podem ser diversos, visto que na matemática existem os mais variados tipos de operações. Apesar dessas características, deseja-se manter as propriedades de fechamento, parcimônia e suficiência, ou seja, deve-se encontrar o menor número possível de operadores capazes de resolver o problema, cujas entradas e saídas sejam compatíveis.

Portanto, visando utilizar o número mínimo de operadores, as primeiras árvores possuíam apenas três: adição, subtração e multiplicação. Esses são suficientes para representar todos números reais caso a escolha de constantes seja adequada, além disso, a saída de cada um deles pode ser a entrada para os demais. Nesse trabalho, as constantes foram escolhidas aleatoriamente no intervalo [-1,1]. Testes preliminares revelaram que eles eram suficientes para conseguir uma representação adequada da função. Foi testada também a divisão protegida e exponencial, para ver se sua inclusão representaria alguma melhoria em relação ao tempo de convergência ou *fitness*, mas seu uso apenas atrapalhou o comportamento das soluções e, portanto, foi abandonado. Com isso foi possível ver que de fato, as três operações básicas eram suficientes.

3.2 Representação

Como a regressão simbólica resulta na criação de uma expressão simbólica que é uma estimativa da função alvo, a representação natural de uma solução é uma árvore de expressão na qual nós internos representam operadores(não terminais) enquanto folhas representam variáveis ou constantes(terminais). As árvores são binárias devido à escolha dos operadores e não necessariamente são balanceadas.

Como essa é uma representação clássica, também são simplificados os demais processos da programação genética, visto que o que funciona com essa representação já é bem definido.

3.3 População inicial

Para a decisão de população inicial, existem 3 métodos comuns: *grow*, que é totalmente aleatório, *full*, que gera árvores balanceadas e *ramped half-half*, que gera tanto árvores aleatórias quanto balanceadas. O funcionamento do método *grow* é o seguinte, dado um nó da árvore, esse nó pode receber aleatoriamente qualquer um dos símbolos usados no problema, de modo que caso o símbolo seja um não-terminal, se repete o processo para seus filhos. Já no método *full*, todos os nós com nível inferior ao tamanho máximo da árvore só podem ser operadores, enquanto os demais só podem ser terminais. Por fim, o método *ramped half-half* funciona da seguinte forma: dado o tamanho máximo(nesse caso 7) de uma solução do problema, a população é dividida em partes iguais para cada possível tamanho de solução(1-7), que são rotuladas com seu respectivo tamanho de solução. Após isso, essas partes

são novamente divididas em duas outras, numa dessas será feito o *grow* com tamanho máximo igual o rótulo e em outra o *full* com tamanho máximo igual o rótulo.

Nesse projeto, foi utilizado o método *ramped half-half*, pois ele garante maior diversidade.

3.4 Seleção

Para esse problema, a seleção foi feita por torneio, para garantir, que melhores indivíduos fossem escolhidos sem ferir a diversidade.

3.5 Operadores Genéticos

No fluxograma podem ser vistos três operadores genéticos: crossover, reprodução e mutação. Porém, para esse trabalho, foram utilizados apenas o crossover e mutação.

O crossover das soluções funciona de uma maneira simples. Dados dois indivíduos, A e B, é selecionada uma subárvore aleatoriamente de cada um deles. Depois disso, checa-se se a subárvore de A cabe no lugar da de B e vice-versa, caso essa condição seja válida é realizada a troca, caso contrário, são selecionadas duas novas subárvores até que essa troca seja possível depois são retornados os dois filhos gerados a partir dessa troca. A forma elitista desse operador é praticamente a mesma, porém, os filhos só são retornados caso sejam melhores que os pais, caso contrário os pais são retornados. Caso um filho seja melhor e o outro pior, é retornado o melhor filho e o melhor pai.

A mutação das soluções é dada pela escolha de um ponto aleatório na árvore que terá seu símbolo trocado por algum outro aleatório. Casos especiais ocorrem quando um símbolo de um tipo(terminal/não-terminal) é trocado por algum de outro tipo. Caso um terminal seja trocado por um não terminal, ocorre uma mutação de expansão [1], na qual são também gerados os novos filhos desse nó. Caso um não terminal seja trocado por um terminal, ocorre uma mutação de redução, na qual os filhos do nó deixam de existir. A forma elitista desse operador só retorna o filho caso ele seja melhor que o pai, retornando o pai em caso contrário.

Como decisão de implementação, as probabilidades de implementação foram feitas complementares de forma que $P_{mutation} = 1 - P_{crossover}$, visando facilitar a etapa experimental do trabalho.

3.6 Fitness

Para o problema de regressão simbólica proposto, a *fitness* de um indivíduo foi definida como a raiz quadrada do erro quadrático médio normalizada (NRMSE), calculada como:

$$fitness(ind) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - EVAL(ind, xi))^2}{\sum_{i=1}^{n}(y_i - \bar{Y})^2}},$$

onde y_i é a saída esperada para a entrada x_i, ind é o indivíduo a ser avaliado, $\bar{Y}$ é a média de todas as saídas, N é o número de instâncias a serem testadas e EVAL calcula a saída de um indivíduo para uma dada instância.

Em um algoritmo genético tradicional como visto na figura 1, o calculo de fitness é feito logo antes da seleção, porém, devido à natureza dos operadores genéticos elitistas propostos, que requerem saber a *fitness* de cada novo individuo tão logo ele é criado, optou-se por avaliar cada indivíduo assim que ele passa a existir ou é modificado, não sendo necessário, portanto, uma etapa separada de avaliação.

Como a *fitness* é uma medida de erro, o problema de regressão simbólica, nesse caso, é um problema de minimização, onde se busca reduzir ao máximo a NMRSE.

4 Experimentos

Para determinação dos parâmetros ótimos, foi utilizada a base de dados synth1, isso, pois ela é menor e, portanto, os testes executam mais rapidamente. Para cada teste, foram executadas 30 repetições com *seeds* diferentes e, os gráficos e tabelas mostram a média dos resultados dessas *seeds*.

O primeiro parâmetro a ser determinado foi o tamanho de população, foram testados os tamanhos 50,100 e 500. Para permitir esse teste, foram usados valores arbitrários dos outros parâmetros: taxa de *crossover* de 0.9, taxa de mutação de 0.1, tamanho de torneio de 2 e 500 gerações.

A figura 2 mostra a média de *fitness* dos melhores individuos desse primeiro experimento.

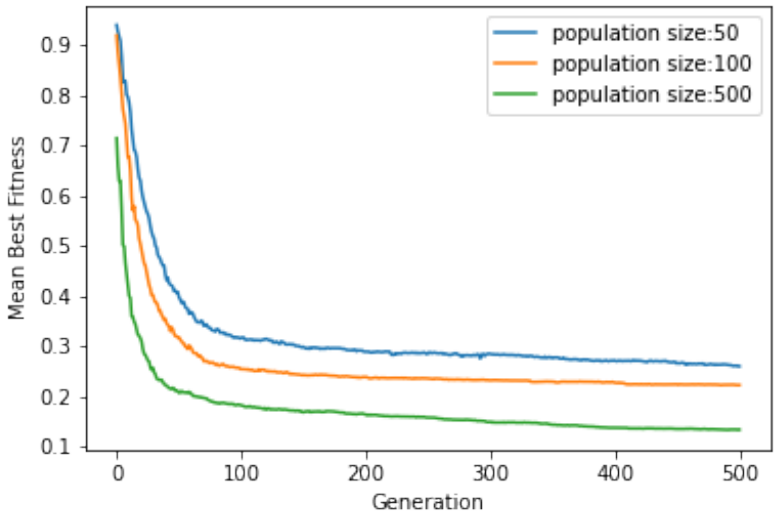


Figura 2: Média de fitness dos melhores indivíduos por tamanho de população

Como esperado, pode-se ver que uma maior população leva a resultados melhores. Isso pode ser muito provavelmente explicado pelo fato de que com uma maior população é esperado que se tenha um maior espalhamento inicial do espaço de busca, reduzindo a chance de convergência em ótimos locais piores. Por meio desse gráfico, pode-se ver também que após 150 gerações aproximadamente, a taxa de variação da *fitness* reduz imensamente. Por isso, para os experimentos subsequentes, foram usadas 150 gerações visando unicamente reduzir o tempo de execução, embora seja evidente que aumentar o número de gerações traz benefícios ao algoritmo. Isso, pois esse número de gerações é suficiente para determinar qual dos parâmetros é melhor. Para a apresentação de resultados serão usadas 500 gerações.

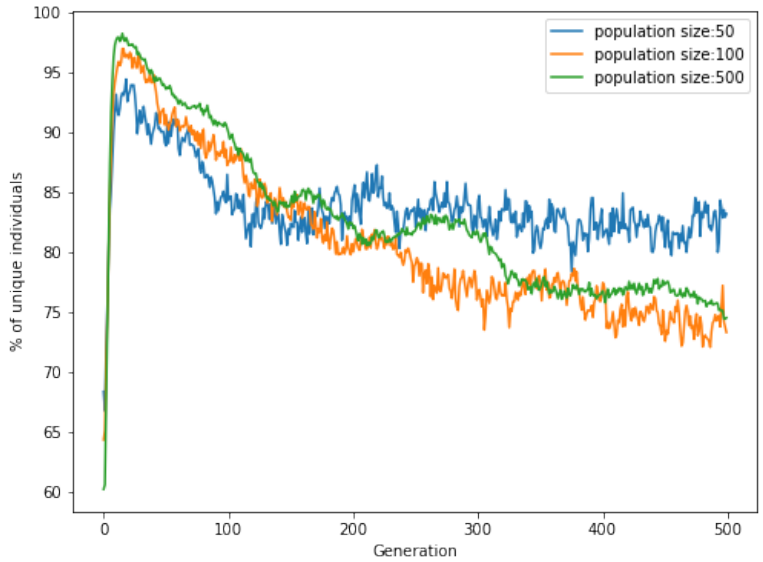


Figura 3: Média de indivíduos únicos(%) por tamanho de população e geração

Quanto à diversidade dos indivíduos ao longo desse processo, é possível ver pela figura 3, que mostra a porcentagem de indivíduos únicos(em média) por geração, que independentemente do tamanho da população, houve certa diversidade em todas gerações. Isso pode ser explicado pelo fato de que apesar da alta taxa de *crossover*, o tamanho do torneio é pequeno e não há nenhuma forma de elitismo. O inicio do processo ter menos diversidade que o final é explicado pelo método de geração da população, visto que o método *ramped half-half* gera muitos indivíduos de tamanho pequeno, que tem, portanto, grande probabilidade de serem iguais, esses, porém, são modificados ao longo das gerações, se tornando únicos. Além disso, pode-se ver que a diversidade cai levemente com o tempo, o que é muito provavelmente consequência do *crossover* e torneio.

Após a definição do tamanho de população, a próxima etapa foi determinar as taxas ideais de mutação e *crossover*, que foram modeladas como complementares, reduzindo o *turning* necessário. A figura 4 mostra a média de *fitness* dos melhores indivíduos desse processo, enquanto a figura 5 mostra a porcentagem de indivíduos únicos por geração para uma dada taxa de mutação e *crossover*. Pode-se perceber pela figura 4 que para esse problema a taxa de mutação mais adequada aparenta ser 0.3, a diferença na média de *fitness* é pequena entre as diferentes taxas, o que pode indicar a necessidade de fazer mais testes para determinar de fato a melhor. Apesar disso, como esperado, a diversidade da população aumenta com uma maior taxa de mutação e diminui conforme o *crossover* aumenta.

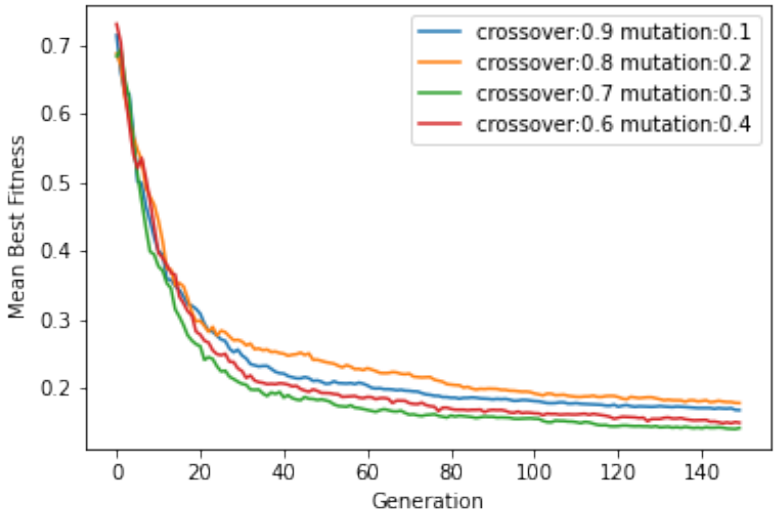


Figura 4: Média de fitness dos melhores indivíduos por taxa de mutação e crossover

A alta diversidade, somada com uma média de *fitness* dos indivíduos(que não são os melhores) alta no geral, são forte indicativo de que está havendo muita exploração(*exploration*) e pouca intensificação(*exploitation*), possivelmente pois o *crossover* em programação genética não mantém o contexto, pode ter efeito destrutivo. Com isso, provavelmente um tamanho maior de torneio é adequado e, além disso, os operadores elitistas discutidos anteriormente podem ser interessantes. Apesar dessa falta de convergência observada, os melhores indivíduos ainda são muito bons e não sujeitos à *overfitting*. Visto que ambos tem relativa diversidade e, todos levaram à boas previsões, optou-se por uma taxa de *crossover* de 0.7 e mutação de 0.3, visto que ela é em média um pouco melhor.

Olhando agora para o tamanho do torneio, a figura 6 mostra a media de melhor *fitness* por tamanho de torneio, enquanto a figura 8 mostra a porcentagem de indivíduos únicos. Como esperado, tamanhos de torneio maiores levam a uma convergência mais acelerada e ao mesmo tempo reduzem a diversidade de indivíduos. Além disso, nota-se que tamanhos de torneio 3,5 e 7 levaram à convergência em uma *fitness* parecida, indicando que apesar de um tamanho de torneio 3 levar a menos intensificação do que os outros, ele ainda foi suficiente para alcançar o mesmo resultado, porém, experimentalmente observou-se overfitting, logo, os tamanhos de torneio 5 e 7 foram mais adequados, por isso, foi escolhido o tamanho de torneio 5. Por fim, a figura 7 mostra a média de *fitness* dos indivíduos por tamanho de torneio, pode-se ver, como esperado, que o maior torneio(7) tem uma variação muito menor de sua

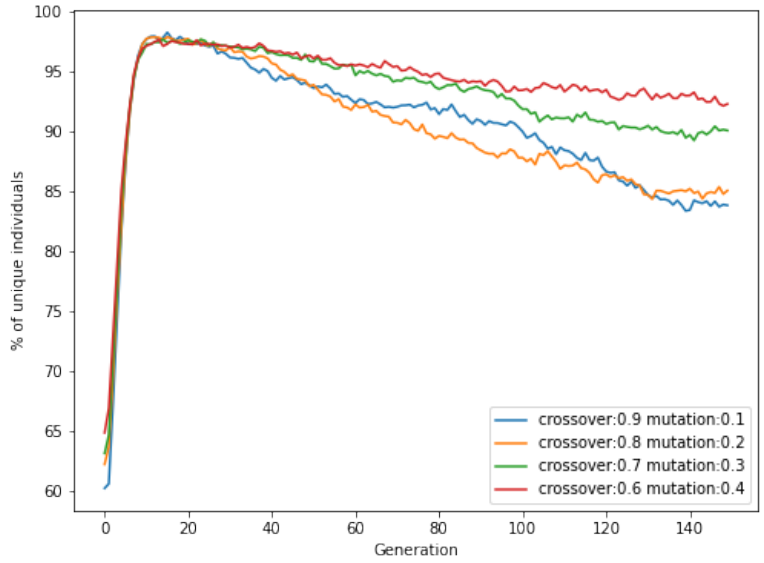


Figura 5: Média de indivíduos únicos(%) por taxa de crossover e mutação

média do que o menor torneio(2), visto que se torna improvável indivíduos ruins serem selecionados.

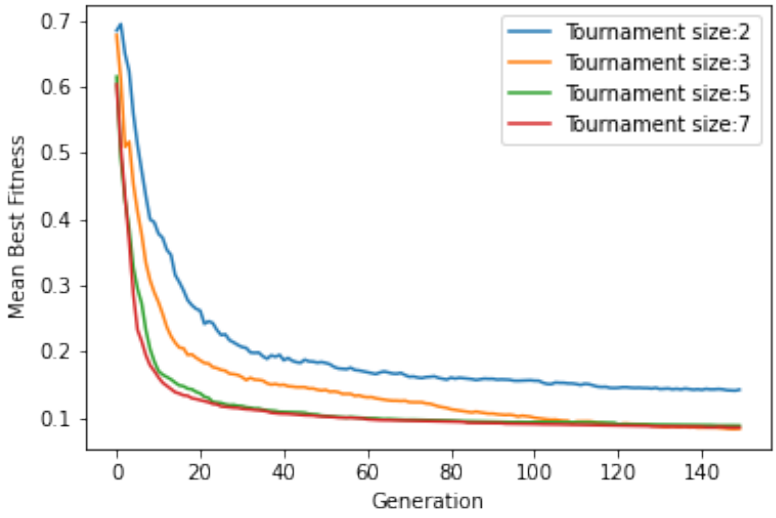


Figura 6: Média de fitness dos melhores indivíduos por taxa de mutação e crossover

Por fim, com os parâmetros encontrados foi testado o operador elitista, visando discernir sua influência no processo. A figura 9 mostra a média da melhor *fitness* por geração enquanto a figura10 mostra a média de todos indivíduos por geração. Como esperado, ao se utilizar operadores elitistas, a *fitness* converge mais rapidamente, em especial, a *fitness* média se torna praticamente igual à melhor *fitness* em menos de 100 gerações, indicando uma possibilidade de convergência prematura. De fato, olhando para o número de indivíduos únicos(15, têm-se que praticamente todos são iguais, o que demonstra que está havendo muita intensificação e pouca exploração. Apesar disso, empiricamente, constatou-se que utilizar o elitismo gerou exatamente a mesma previsão para a base synth1, o que significa que provavelmente ambos modelos chegaram a uma mesma árvore de expressão, a figura 12 e

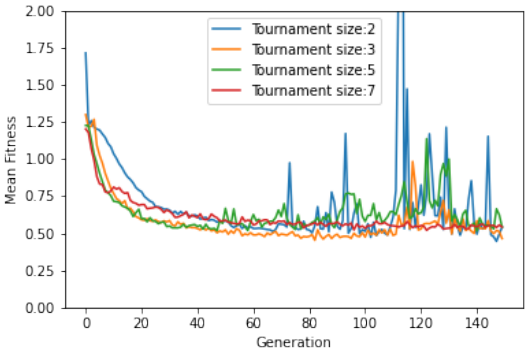


Figura 7: Fitness media por tamanho de torneio

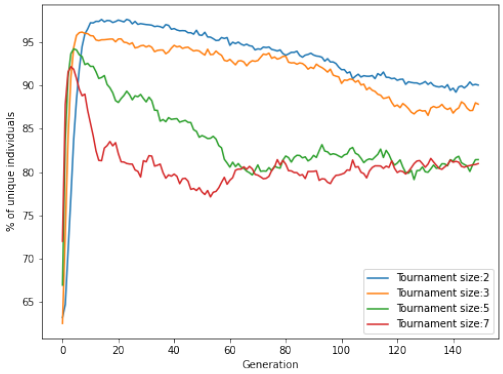


Figura 8: Média de indivíduos únicos(%) por torneio

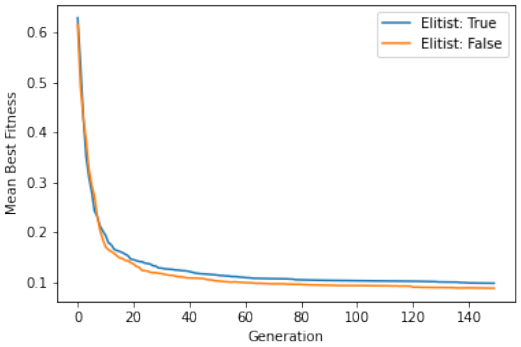


Figura 9: Média de fitness dos melhores indivíduos com ou sem operador elitista

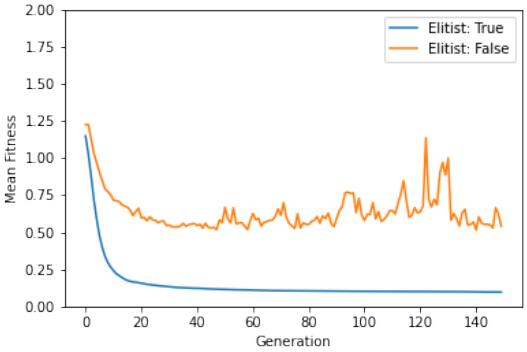
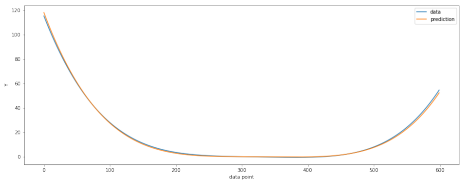
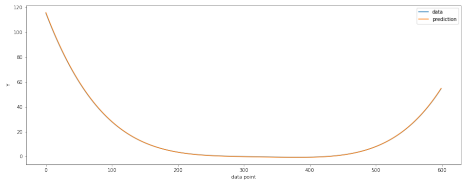


Figura 10: Média de fitness de todos indivíduos com ou sem operador elitista

a tabela **??** mostram isso.



(a) Função estimada para a base synth1 com elitismo



(b) Função estimada para a base synth1 sem elitismo

Figura 12: Previsões com ou sem operadores elitistas. O eixo x representa o ponto da base de dados a ser estimado e o eixo y representa o valor.

Base de dados	elitism	no elitism:
synth1	0.0328	0.0328
synth2	0.612	0.358
concrete	0.558	0.551

Tabela 1: Erros para as diferentes bases de dados

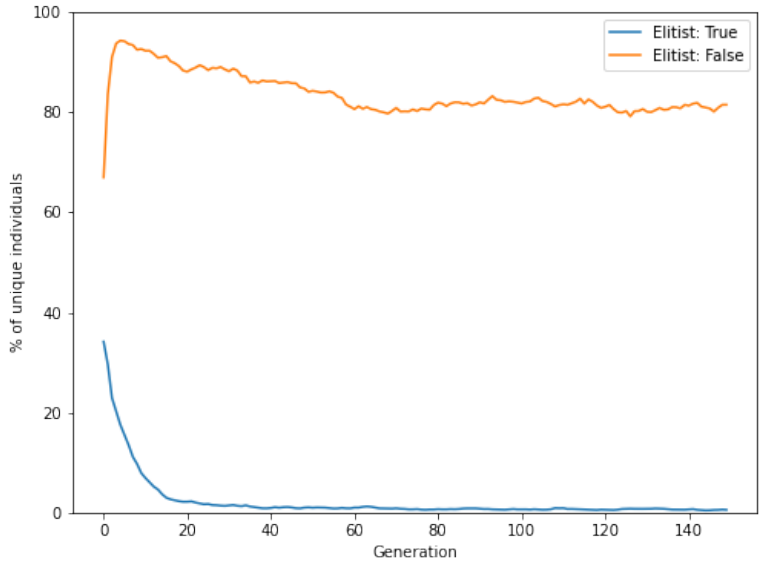


Figura 11: Média de indivíduos únicos(%) com ou sem operador elitista.

5 Resultados

Encontrados os parâmetros aparentemente ótimos de tamanho de população (500), taxa de mutação(0.3), taxa de *crossover*(0.7), tamanho de torneio(5) e não utilização de operadores elitistas, a próxima etapa foi testar o modelo resultante em cada uma das bases de dados de interesse. As estimativas do modelo para cada base de dados podem ser vistas abaixo:

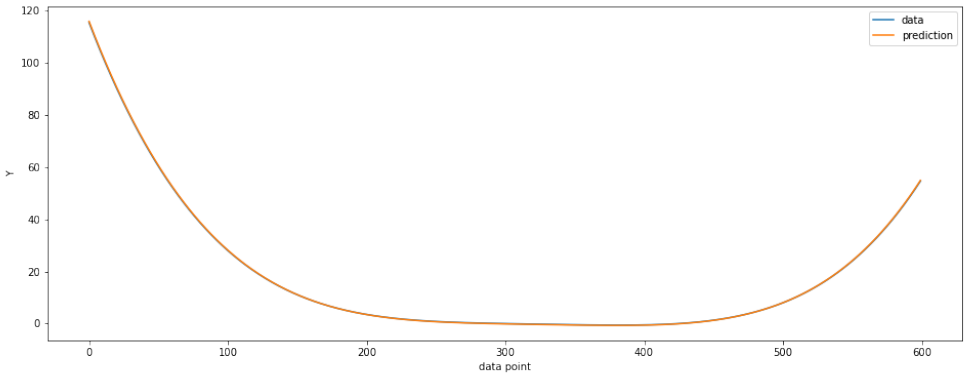


Figura 13: Função estimada para a base synth1

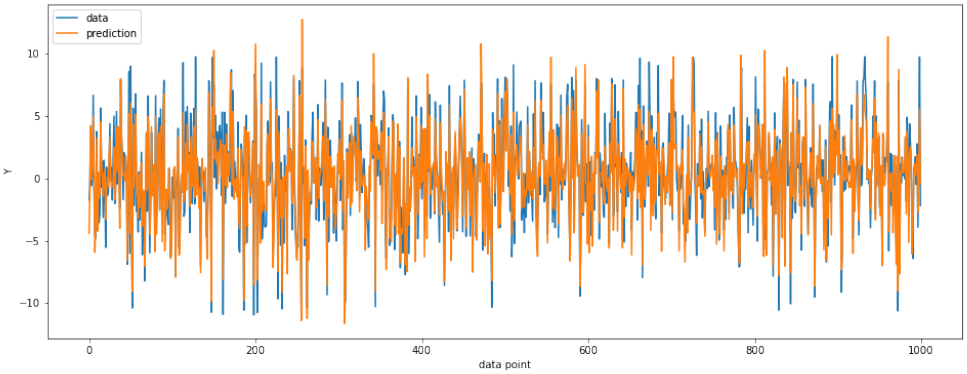


Figura 14: Função estimada para a base synth2

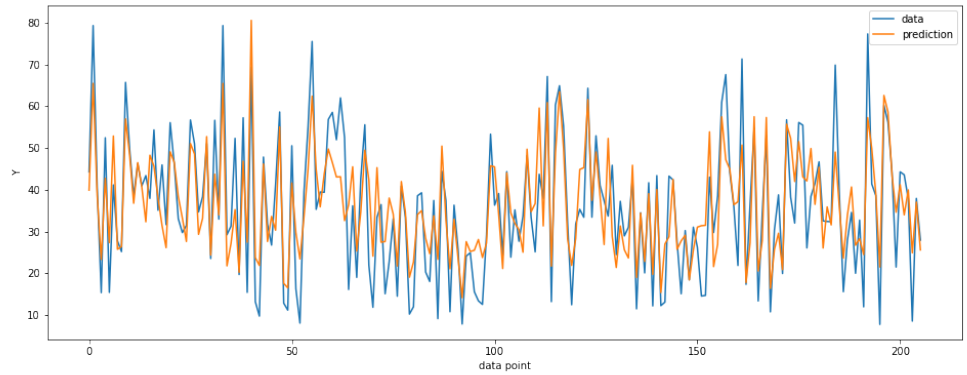


Figura 15: Função estimada para a base concrete

Pode-se ver que há certa semelhança das estimativas para todas as funções. Porém, apesar disso, apenas a estimativa para a base synth1 é realmente muito próxima da real, as demais apenas tem certa semelhança às tendencias das curvas.

6 Conclusão

Nesse trabalho foi possível explorar todo o processo de criação de um algoritmo de programação genética, desde a definição de operadores até a testagem de parâmetros e previsões. Por meio de experimentos, foi possível determinar parâmetros que levaram à resultados satisfatórios, com previsões muito próximas da função real de interesse para a base synth1, embora para as demais bases de dados, as previsões foram piores, conseguindo capturar apenas a tendência dos dados de certa maneira.

Foi também explorada a influência de cada parâmetro na velocidade de convergência e diversidade de população, sendo possível observar vários dos aspectos discutidos em sala. Por fim, o algoritmo foi construído observando as propriedades de fechamento, parcimônia e suficiência.

Uma das limitações do algoritmo implementado se dá na velocidade de execução, consequência da escolha de linguagem Python, que é até 45 vezes mais lenta que a linguagem C [2]. A utilização de uma linguagem mais rápida facilitaria consideravelmente o processo de tuning dos parâmetros, visto que cada parâmetro demorou horas para ser testado em todas suas variações em 30 diferentes sementes. Caso o programa tivesse sido escrito em uma linguagem mais eficiente, ficaria mais fácil testar operadores alternativos, como por exemplo logaritmos.

Referências

[1] Gisele L. Papa. Slides de programação genética, 2022. Accesso em: 13-05-2022.

[2] Peter Xie. How slow is python compared to c. https://peter-jp-xie.medium.com/how-slow-is-python-compared-to-c-3795071ce82a#:~:text=It%20is%20450%20million%20loops,mode%20for%20a%20better%20performance. Accessed: 15-05-2022.