Maximização de função
Trabalho 03

Computação Evolutiva
at Universidade Federal de Uberlândia

Antonio Fernandes Valadares

28 de dezembro de 2021

Número de matrícula: 11711ECP015
Professor: Keiji Yamanaka

Objetivo

O objetivo desse trabalho é realizar a maximização de uma função com duas variáveis utilizando algoritmos genéticos, a função a ser maximizada é a seguinte:

$f(x, y) = 10 + xsen(4x) + 3sen(2y)$

Onde: $0 \leq x \leq 4$ e $0 \leq y \leq 2$

Como requisitos do trabalho é necessário usar o Elitismo e realizar a seleção por torneio.

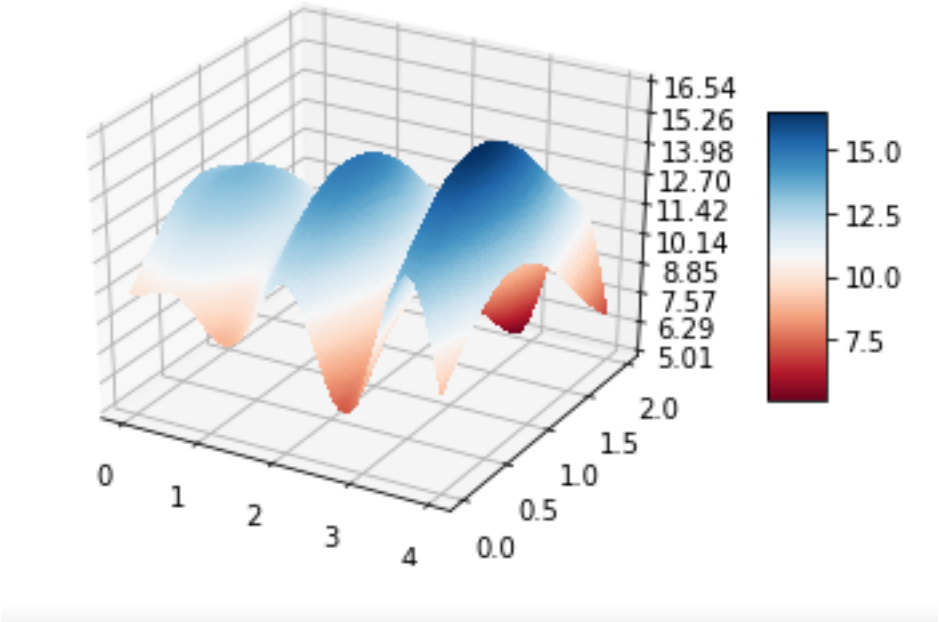

Figura 1: Gráfico da função a ser maximizada

Construção do algoritmo

O algortimo em si consiste de quatro partes, primeiro é necessário gerar uma população estocástica inicial, segundo calculamos a aptidão de cada indivíduo, após isso realizamos o cruzamento utilizando as técnicas de elitismo e torneio e depois aplicamos uma chance de mutação para cada indivíduo, com isso geramos uma nova população. Esse processo é repetido por um número determinado de gerações.

O código foi todo desenvolvido em python, utilizando notebooks. Para melhor visualização do processo, escolhi utilizar a biblioteca pandas do python, dessa forma a população estaria armazenada em um objeto do tipo DataFrame, que possui formato tabular e facilita a visualização dos dados. A função que gera a população inicial recebe um parâmetro size, que define o tamanho da população. É gerado dois vetores aleatório do tamanho da size, um de números entre 0 e 4 e um de número entre 0 e 2, para x e y, respectivamente.

Para gerar o cromossomo de x foi utilizado 10 genes, já para o y, por se tratar de um dominio menor foram utilizados 9 genes. Logo geramos uma string de 10 bits que representa o cromossomo de x e uma string de 9 bits que representa o cromossomo y. Dessa forma temos uma precisão de mais ou menos de 0.002 para x e y.

	x	y	bin_x	bin_y
0	2.771553	0.728413	1011000101	010111010
1	0.590972	0.279750	0010010111	001000111
2	1.095496	1.653934	0100011000	110100111
3	0.938333	0.253081	0011110000	001000000
4	3.163297	1.836285	1100101001	111010110
5	3.864312	0.851033	1111011101	011011001
6	1.785600	0.500030	0111001001	010000000
7	3.789753	1.521795	1111001010	110000101
8	1.846303	0.111509	0111011000	000011100
9	0.822442	1.324531	0011010010	101010011

Figura 2: Exemplo de população de 10 indivíduos gerada estocasticamente pelo algoritmo

Cálculo da aptidão

A função que calcula o $fitness$, recebe uma população e aplica a função que estamos tentando maximizar como função objetivo para os valores de x e y de

cada registro. Então ele acrescenta uma coluna no nosso DataFrame que indica a aptidão de cada indivíduo. A função também ordena os indivíduos por sua aptidão e exibe o melhor indivíduo da geração.

```
Melhor resultado:
         x         y       bin_x      bin_y   fitness
1.868075  0.555299  0111011110  010001110  14.42154
```

	x	y	bin_x	bin_y	fitness
1	1.868075	0.555299	0111011110	010001110	14.421540
7	3.775695	1.240713	1111000110	100111101	13.987783
2	3.701109	0.138502	1110110011	000100011	13.727701
3	1.520728	0.968669	0110000101	011110111	12.498188
8	3.118247	1.125251	1100011110	100100000	12.042505
6	0.390939	0.218463	0001100100	000110111	11.660396
0	2.489659	0.471412	1001111101	001111000	11.160780
5	3.963783	1.303409	1111110110	101001101	10.947708
9	2.912406	0.632617	1011101001	010100001	10.549671
4	0.810377	1.632962	0011001111	110100010	9.547133

Figura 3: Cálculo do fitness para uma população de 10 indivíduos

Crossover

A função responsável por fazer o crossover dos indivíduos e gerar uma nova população recebe como parâmetros a taxa de crossover, o tamanho da nova população e um valor de k, que é para realizar o torneio. Como o elitismo e a seleção por torneio são requisitos do trabalho, o primeiro indivíduo colocado na nova população é o que obteve o melhor fitness entre os indíviduos da população anterior.

Após isso o torneio é realizado duas vezes, para selecionar dois pais. A técina de seleção por torneio consiste em sortear k indivíduos da população anterior e selecionar o de maior aptidão para realizar o crossover. Com os dois pais selecionados utilizando o torneio, é escolhido um número aleatório de 0 a 1, caso o número seja maior que a taxa de crossover os pais são passados a frente, caso contrário são gerados filhos e os filhos são passados para a nova população. Os filhos são gerados cruzando o material genético dos seus pais, tanto para x

quanto para y. Esse processo é repetido até que a população atinja a quantidade de indivíduos desejados.

	x	y	bin_x	bin_y	fitness
0	3.511813	1.146978	1110000011	100100101	15.746783
1	3.511719	1.144531	1110000011	100100101	0.000000
2	3.511719	1.144531	1110000011	100100101	0.000000
3	3.511813	1.146978	1110000011	100100101	15.746783
4	1.597137	0.852042	0110011000	011011010	13.141360
5	3.351562	1.402344	1101011010	101100111	0.000000
6	3.511719	1.671875	1110000011	110101100	0.000000
7	1.593750	0.859375	0110011000	011011100	0.000000
8	1.996094	0.101562	0111111111	000011010	0.000000
2	2.269966	0.652003	1001000101	010100110	13.661374

Figura 4: Nova população de 10 indivíduos geradas por crossover

Nesse caso os parâmetros utilizados foram: $size = 10$, $crossoverrate = 0.6$ e $k = 3$. Os indivíduos que possuem fitness igual a 0 são filhos. O indivíduo número 0 foi o melhor indivíduo da geração anterior.

Mutação

A função responsável pela mutação recebe uma população e uma taxa de mutação. Ela aplica essa taxa de mutação para cada indivíduo, sorteando um número entre 0 e 1, caso seja menor que a taxa de mutação escolhida o indivíduo sofre mutação e um bit aleatório é trocado.

	x	y	bin_x	bin_y	bin_x_mut	bin_y_mut
0	2.292801	1.747067	1001001010	110111111	1001000010	110111111
1	3.541508	0.397045	1110001010	001100101	0110001010	001100001
2	0.292373	1.357324	0001001010	101011011	0001011010	100011011
3	3.570932	1.133728	1110010010	100100010	0110010010	101100010
4	2.516610	0.796921	1010000100	011001100	1010000101	011001100
5	3.374733	0.910475	1101011111	011101001	1101001111	011101011
6	3.554108	1.186712	1110001101	100101111	0110001101	100101111
7	3.896652	0.367526	1111100101	001011110	0111100101	001011110
8	2.131763	0.945203	1000100001	011110001	1000100001	011110011
9	3.850400	1.753715	1111011001	111000000	1111011001	111100000

Figura 5: População de 10 indivíduos que sofreu mutação

Mutação realizada para uma população de 10 indivíduos para uma taxa de mutação de 0.5. Foi acrescentado uma coluna com o código binário mutado para melhor visualização. Podemos notar que os indivíduos 1, 2, 3, 4, 5, 6, 7 sofreram mutações, pois um bit está diferente.

Testes e Resultados

Foram realizados vários testes, com vários parâmetros diferentes. Por se tratar de um problema teoricamento simples não é necessário uma população muito grande nem muitas gerações para obter resultados interessantes. Em geral uma população de 20 indivíduos e 20 gerações geram resultados bons, próximos a 16.54 que é muito perto do máximo da função.

Em relação ao parâmetro k, para realizar o torneio. Os testes mostraram que valores muito altos podem fazer com que o indivíduo mais apto domine a população, dessa forma dificultando geração de resultados melhores nas próximas gerações, o valor de k=3 foi o mais testado e que pareceu mais adequado. Também não é interessante deixar uma taxa de mutação muito alta, pois apesar disso dificultar que um indíviduo domine a população, faz com que a média de fitness não se mantenha tão boa, e também pode ocasionar a perda de bons indivíduos, em geral a taxa de mutação de 10% entregou bons resultados. Em relação a taxa de

crossover, os melhores resultados foi entre 0.6 e 0.8, taxas de crossover muito altas dificultam que a média de aptidão da população aumente, mas taxas pequenas fazem com que o algoritmo demore mais para achar um indivíduo ótimo.

O melhor resultado encontrado foi o valor de 16.54306, que é o resultado da função para os valores de $x = 3.550781$ e $y = 0.785156$.

```
 Resultado final !!!!

Melhor resultado:
         x         y       bin_x      bin_y    fitness
 3.511719  0.789062  1010000011  011001010  16.497333

Média de aptidão:  14.688867018469839
```

Figura 6: Resultado de uma execução com os seguintes parâmetros: size=20, crossover_rate=0.6, k=3, geracoes=20, mutation_rate=0.1

Podemos também visualizar o gráfico de desempenho para essa execução.

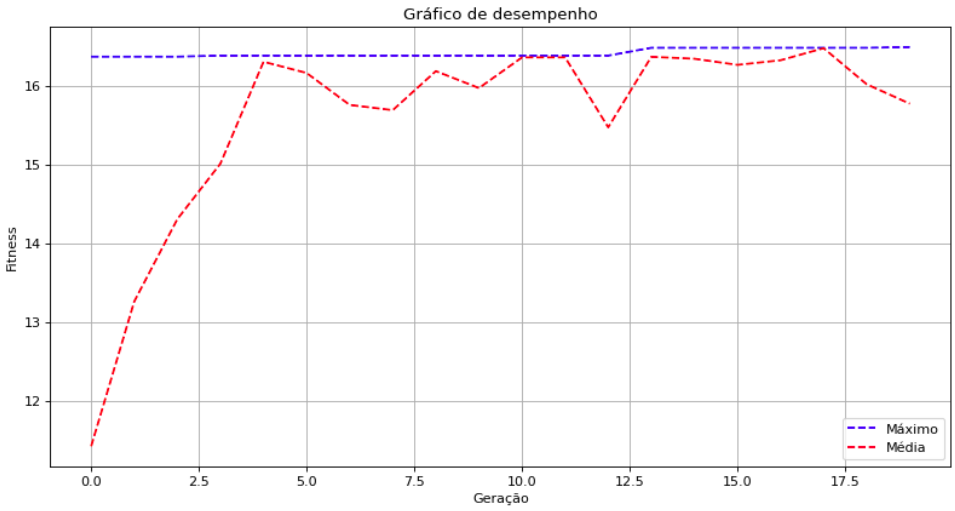

Figura 7: Gráfico de desempenho de uma execução com os seguintes parâmetros: size=20, crossover_rate=0.6, k=3, geracoes=20, mutation_rate=0.1

Oberservamos que a média da aptidão das gerações tende a convergir para o melhor indivíduo, em algumas pontos a média piora por conta de mutações ou cruzamento que gerem indivíduos não tão interessantes.

Para averiguar se o algoritmo estava tendo bons resultados no geral, realizei 30 testes e plotei a distribuição dos seus resultados. Utilizandos os mesmo parâmetros utilizados nos testes acima.

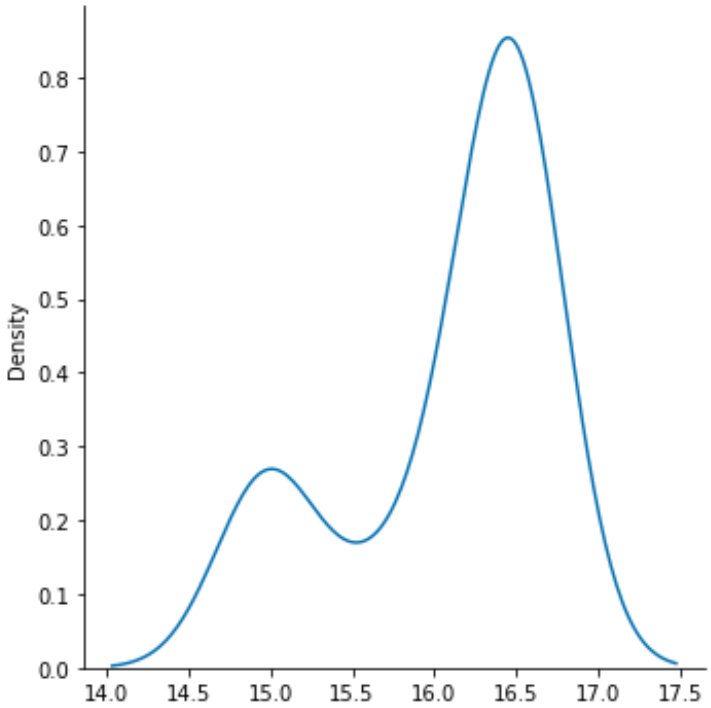

Figura 8: Distribuição de resultados para 30 execuções do algoritmo, com os parâmetros: size=20, crossover_rate=0.6, k=3, geracoes=10, mutation_rate=0.1

É possível ver que em grande maioria das execuções o algoritmo obteve bons resultados, ou seja, próximo de 16,54. Porém é possível perceber também que algumas execuções se perderam em resultados por volta de 15, o que me levou a acreditar que esse valor se trata de um máximo local e o algoritmo pode acabar se perdendo nesses locais as vezes. Uma solução é rodar o algoritmo por mais gerações para se certificar que ele irá achar um resultado bom.