

عنوان پروژه :

تقریب تابع با کمک برنامه نویسی ژنتیک (GP)

آیسا میاهی نیا 99522149

دکتر عبدی

در این پروژه همانطور که می دانیم یک سری نقاط به عنوان ورودی داریم و مقدار خروجی آن ها را به ازای تابعی که داخل black box می باشد نیز داریم. برای اینکه بتوانیم این نقاط ورودی و خروجی را داشته باشیم، با استفاده از یک تابع به نام csv_creator که در فایل creat_input.py می باشد، به تعدادی که مورد نظرمان است نقاطی رندوم را تولید کرده و خروجی آن تابعی که در black box می باشد را به ازای آن ورودی حساب می کنیم. و در فایل CSV ذخیره می کنیم.

تابع اصلی که برای اجرای نسل ها می باشد در فایل driver.py به نام تابع GeneticAlgorithm می باشد که ورودی هایی که می گیرد یکی تعداد variable هایی که تابع مدنظرمان دارد و اسم فایل CSV که نقاط ورودی خروجی مان دارد می باشد.

در این تابع ابتدا به تعداد 100 تا جمعیت رندوم از یک سری تابع رندوم که تعداد متغیرهای آن ها به تعداد ورودی ای که دادیم می باشد، می سازیم. سپس fitness_function هر یک را حساب می کنیم. Fitness_function را MSE در نظر می گیریم. و با استفاده از نقاط ورودی و خروجی و محاسبه تابع هایی که در نسل صفرم هستند به ازای آن ورودی ها مقدار هر یک را محاسبه می کنیم. سپس fitness های محاسبه شده را به همراه خود عبارت به صورت یک تاپل داخل یک لیست قرار می دهیم و آن را براساس fitnessشان سورت میکنیم.

فرض می کنیم که 20 تا نسل داریم و پس از 20 نسل جواب باید پیدا شود(هرچند از آن جایی که رندوم است ممکن با یکبار ران کردن جواب پیدا نشود و در بار دوم یا سوم جواب یافت شود) پس یک for داریم که 20 بار اجرا می شود و هر 20 بار کارهای زیر را بر روی نسل انجام می دهد تا به نسل جدیدی برسد.

ابتدا قبل از اینکه عملیات های crossover و mutation را انجام دهد، چک می کند که اگر fitness ما از یک مقداری(در اینجا 150) کمتر بود یعنی ما به جواب رسیدیم و جواب را با fitness اش نمایش دهد. ولی از آن جایی که ممکنه به جواب های بهتری در نسل های بعدی برسیم به روندش ادامه میده تا 20 نسل کامل بشود.

از آن جایی که ممکن است با انجام crossover و mutation یک سری جواب احتمالی خوب را که تا حدی به جواب اصلی مسئله نزدیک می باشد از دست بدهیم پس هر بار 20 درصد بهترین جواب را ذخیره می کنیم تا در نسل بعدی هم جزو جمعیتمان باشد.

برای انتخاب کردن هر دو parent و اعمال crossover (selection)، روش ها و الگوریتم های متفاوتی وجود دارد. در ابتدا به طور کاملا رندونم این parent ها را انتخاب می کردم که این روش اصلا بهینه نبود و نمیتوانست جواب را با تقریب خوبی تا نسل 20 ام پیدا کند به همین دلیل از روش roulette wheel استفاده کردم به این صورت که مجموع fitness ها را محاسبه کردم و هر fitness را بر حاصل جمع تقسیم کردیم و از یک کم کردم و به هر تابع این مقدار را اختصاص دادم.

پس از استفاده از این روش و ذخیره 20 درصد بهینه هر نسل توانستم تا حد خوبی کارایی الگوریتم را افزایش دهم.

برای crossover دو parent انتخاب شده، دو عدد رندوم که در بازه طول کوتاه ترین عبارت می باشد پیدا کردم و اگر آن operation نبود به پیدا کردن آن اعداد ادامه دادم پس از پیدا شدن از آن قسمت عبارات را تیکه کردم و به یک دیگر وصل کردم تا دو عبارت جدید داشته باشیم.

شرط خاتمه الگوریتم هم همانطور که پیش تر هم گفته شده بعد از 20 نسل می باشد که البته میتوانست شرط خاتمه پیدا کردن جواب بهینه با تقریب خوبی باشد.

در این پروژه میتوان توابعی را که دارای 3 متغیر و عملیات های جمع و ضرب و تفریق و تقسیم و توان می باشند را محاسبه کرد.

یکی از مشکلاتی که در پیاده سازی این الگوریتم داشتم، تقسیم بر صفر بود. چون از تابع eval برای محاسبه مقدار توابع استفاده می کردم وقتی بر صفر تقسیم می شد exception میداد که با try و catch این مشکل برطرف کردم.

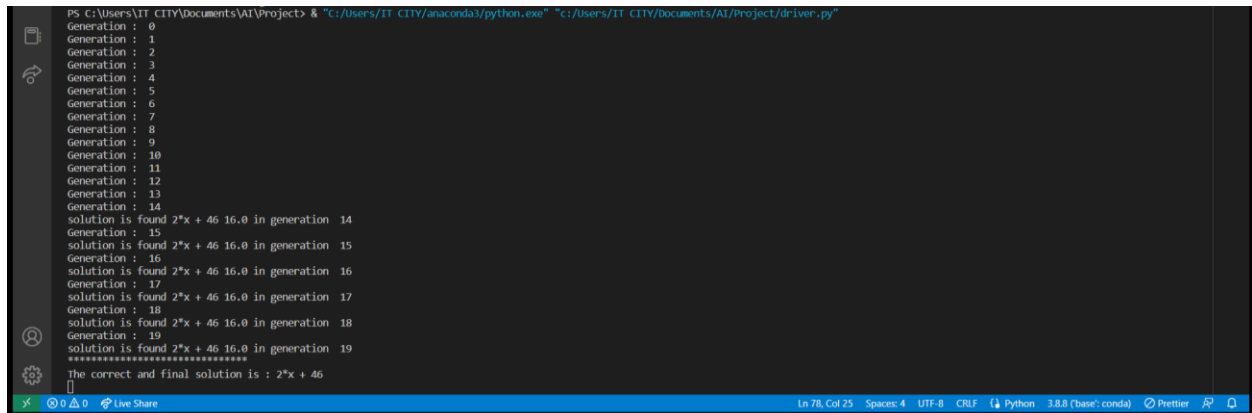
برای ران کردن برنامه کافیس، فایل driver.py را اجرا کنید و تعداد متغیر ها و تابع موردنظر خود را به عنوان ورودی در ترمینال وارد نمایید.(ممکن است با یکبار ران کردن در 20 نسل به جواب نرسید به ران کردن خود ادامه دهید :)

آزمایش های انجام شده:

1. **آزمایش اول** : تابع مجهول، تابعی تک متغیره و دارای عملیاتی است که این کد آن را

پشتیبانی می کند.

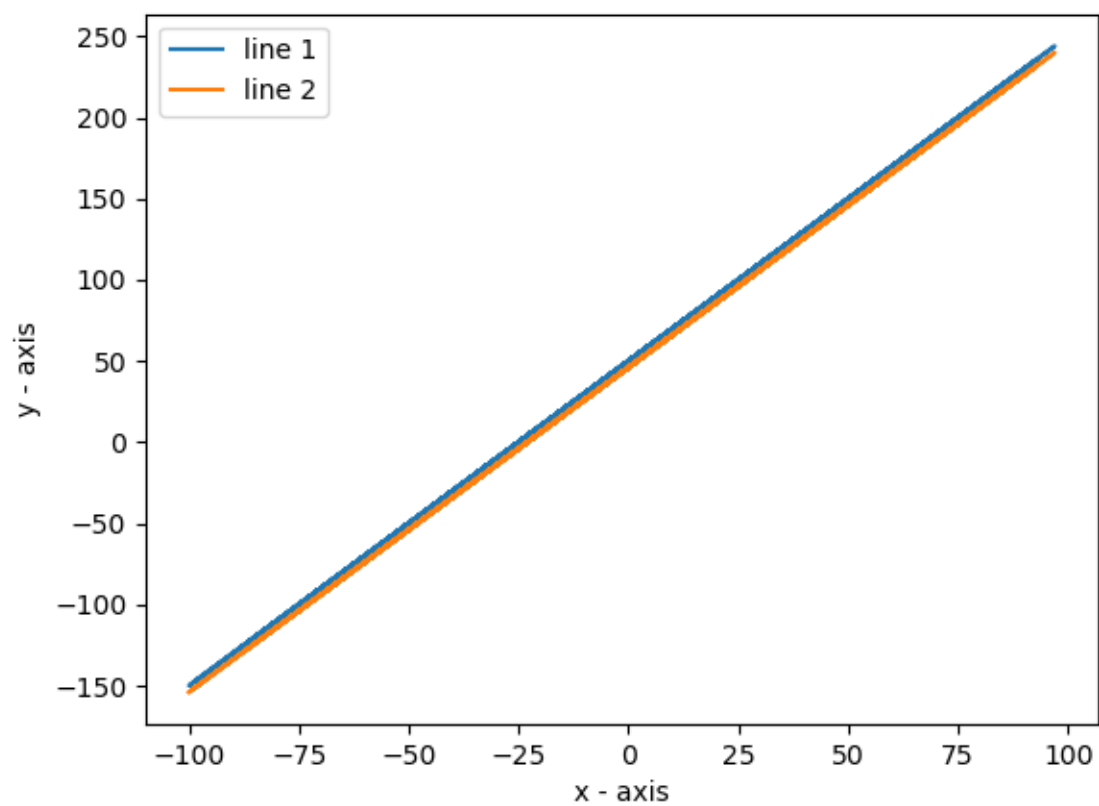
$$F(x) = 2x + 50$$



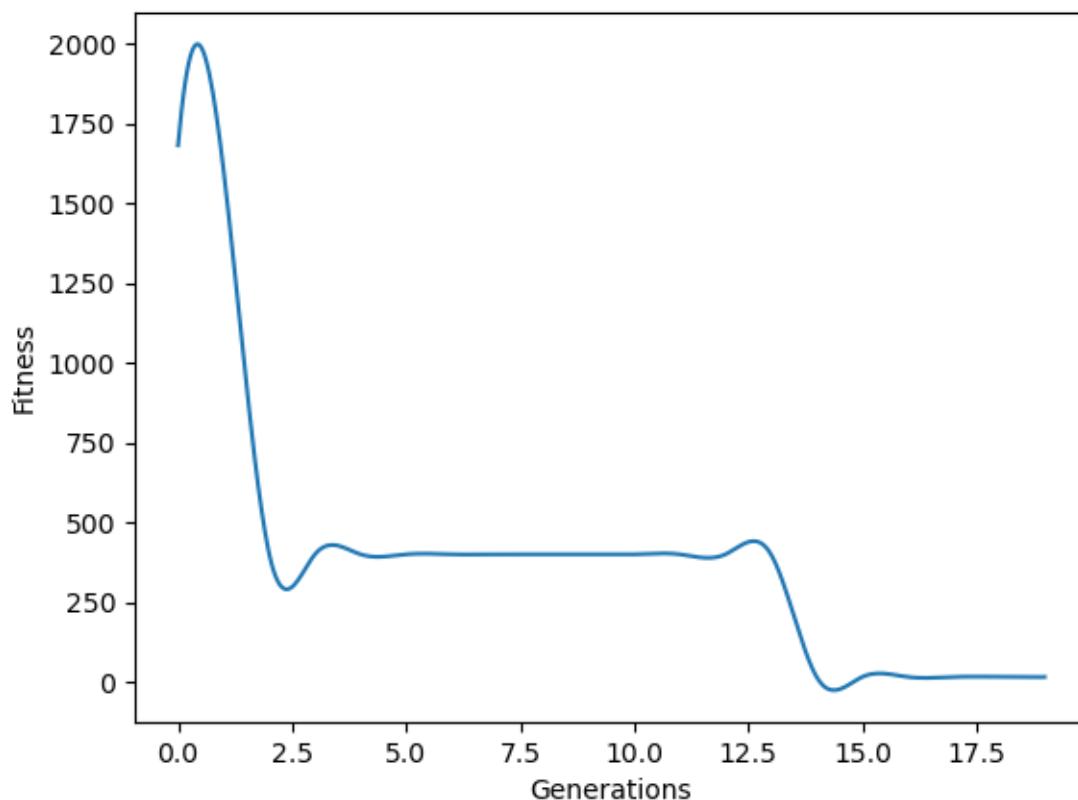
```
PS C:\Users\IT CITY\Documents\AI\Project> & "c:/Users/IT CITY/anaconda3/python.exe" "c:/Users/IT CITY/Documents/AI/Project/driver.py"
Generation : 0
Generation : 1
Generation : 2
Generation : 3
Generation : 4
Generation : 5
Generation : 6
Generation : 7
Generation : 8
Generation : 9
Generation : 10
Generation : 11
Generation : 12
Generation : 13
Generation : 14
solution is found 2*x + 46 16.0 in generation 14
Generation : 15
solution is found 2*x + 46 16.0 in generation 15
Generation : 16
solution is found 2*x + 46 16.0 in generation 16
Generation : 17
solution is found 2*x + 46 16.0 in generation 17
Generation : 18
solution is found 2*x + 46 16.0 in generation 18
Generation : 19
solution is found 2*x + 46 16.0 in generation 19
*****
The correct and final solution is : 2*x + 46
```

شکل 1- خروجی کد به ازای ورودی $2x+50$

همانطور که در شکل 1 مشاهده می کنید تابع برای اولین بار در نسل 14 با fitness 16 به مقدار $2x+46$ پیدا شده است. نمودار تابع اصلی (رنگ آبی) و تابع پیدا شده (رنگ قرمز) هم در شکل 2 قابل مشاهده است که میتوان دید تقریباً بر هم منطبق هستند.



شکل 2- نمودار تابع $2x+46$ و $2x+50$



شکل 3- نمودار تغییر fitness_function به ازای هر نسل

2. **آزمایش دوم:** تابعی را در نظر میگیریم که دارای عملیاتی است که این کد آن را پشتیبانی نمی کند.

$$F(x) = \sin(x)$$

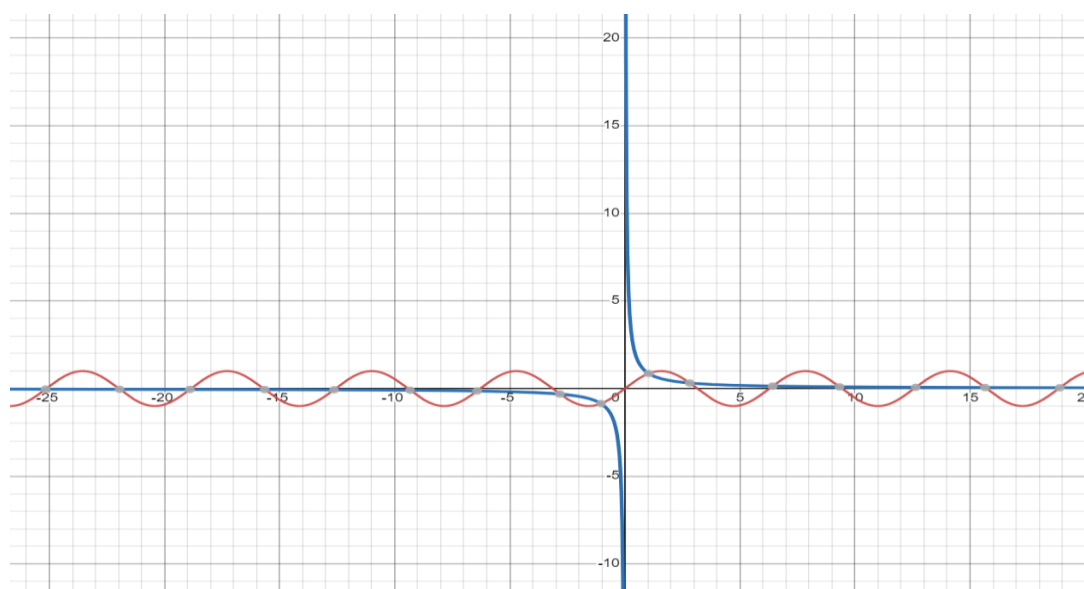
```

Generation : 0
solution is found 1/x 0.47724791280119143 in generation 0
Generation : 1
solution is found 1/x 0.47724791280119143 in generation 1
Generation : 2
solution is found 7/(9*x) 0.47677572174066496 in generation 2
Generation : 3
solution is found 7/(9*x) 0.47677572174066496 in generation 3
Generation : 4
solution is found 7/(9*x) 0.47677572174066496 in generation 4
Generation : 5
solution is found 7/(9*x) 0.47677572174066496 in generation 5
Generation : 6
solution is found 7/(9*x) 0.47677572174066496 in generation 6
Generation : 7
solution is found 7/(9*x) 0.47677572174066496 in generation 7
Generation : 8
solution is found 7/(9*x) 0.47677572174066496 in generation 8
Generation : 9
solution is found 8/(9*x) 0.4766491975949671 in generation 9
Generation : 10
solution is found 8/(9*x) 0.4766491975949671 in generation 10
Generation : 11
solution is found 8/(9*x) 0.4766491975949671 in generation 11
Generation : 12
solution is found 8/(9*x) 0.4766491975949671 in generation 12
Generation : 13
solution is found 8/(9*x) 0.4766491975949671 in generation 13
Generation : 14
solution is found 8/(9*x) 0.4766491975949671 in generation 14
Generation : 15
solution is found 8/(9*x) 0.4766491975949671 in generation 15
Generation : 16
solution is found 8/(9*x) 0.4766491975949671 in generation 16
Generation : 17
solution is found 8/(9*x) 0.4766491975949671 in generation 17
Generation : 18
solution is found 8/(9*x) 0.4766491975949671 in generation 18
Generation : 19
solution is found 8/(9*x) 0.4766491975949671 in generation 19

```

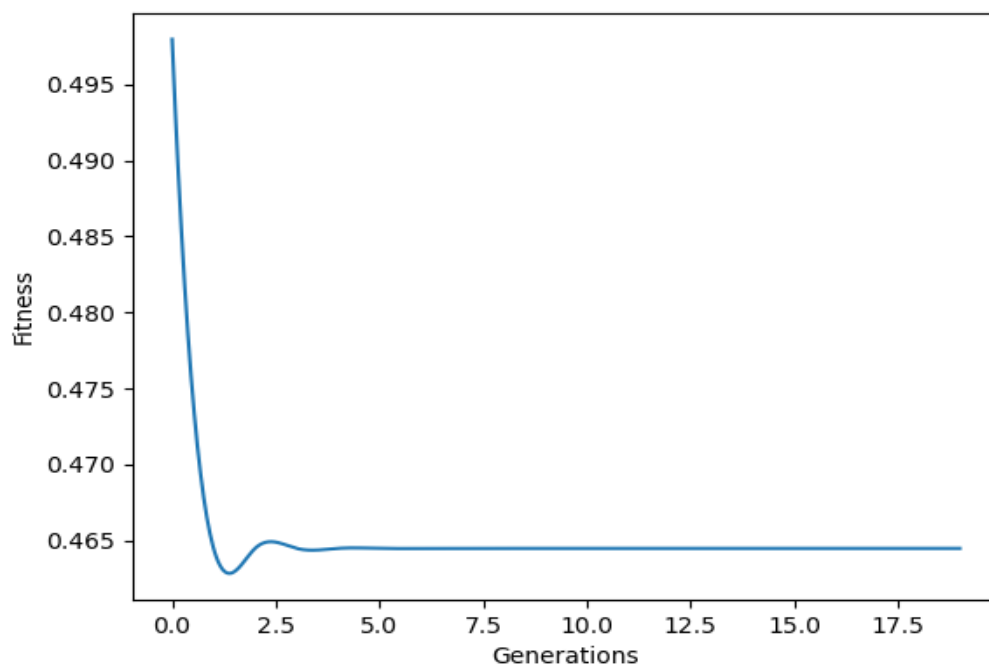
شکل 4 - خروجی کد به ازای تابع $\sin(x)$

همانطور که در شکل 4 مشاهده می کنید، خروجی به ازای تابع $\sin(x)$ ، تابع $8/9*x$ می باشد که با fitness_function ، 0.46 پیدا شده است و نمودار های هر دو تابع را می توانید در شکل 5 مشاهده کنید.



شکل 5 – نمودار توابع $\sin(x)$ و $8/9*x$

با توجه به اینکه این کد، تابع \sin را جزو عملیات های خود در نظر نمیگیرد پس مقدار خروجی را براساس عملیاتی هایی می دهد که می شناسد و تابع خروجی را با توجه به مقدار های ورودی در آن بازه مورد نظر به دست می آورد. یعنی شاید بتوان گفت در آن بازه این دو نمودار شبیه بهم رفتار می کنند.



شکل 6 – نمودار تغییر fitness_function به ازای هر نسل

3. **آزمایش سوم:** در این آزمایش تابع ما دارای دو متغیر x , y می باشد.

$$F(x,y) = 2x + y + 5$$

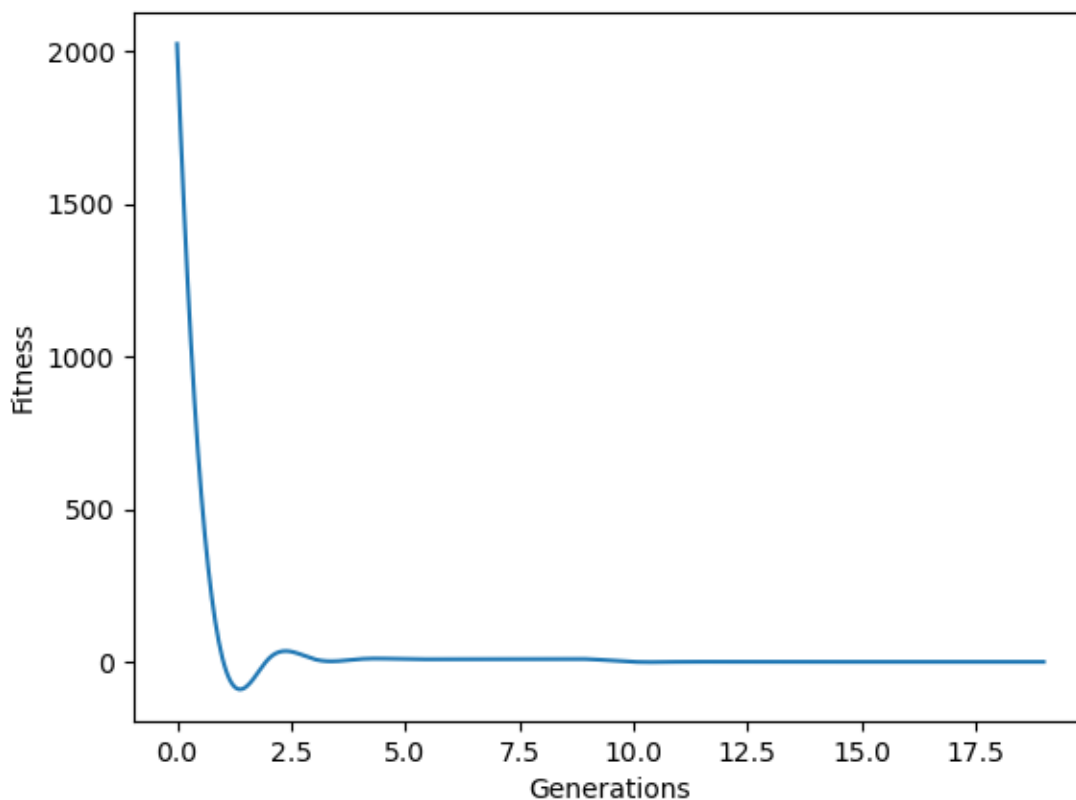
همانطور که در شکل 7 مشاهده می کنید، خروجی کد دقیقا همان تابع ما می باشد و با fitness

صفر به دست آمده است به همین دلیل نمودار آن هم دقیقا بر خودش منطبق می شود.

نمودار fitness_function براساس نسل آن هم در شکل 8 آمده است.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Generation : 9
solution is found 2*x + y + 2 9.0 in generation 9
Generation : 10
solution is found 2*x + y + 5 0.0 in generation 10
Generation : 11
solution is found 2*x + y + 5 0.0 in generation 11
Generation : 12
solution is found 2*x + y + 5 0.0 in generation 12
Generation : 13
solution is found 2*x + y + 5 0.0 in generation 13
Generation : 14
solution is found 2*x + y + 5 0.0 in generation 14
Generation : 15
solution is found 2*x + y + 5 0.0 in generation 15
Generation : 16
solution is found 2*x + y + 5 0.0 in generation 16
Generation : 17
solution is found 2*x + y + 5 0.0 in generation 17
Generation : 18
solution is found 2*x + y + 5 0.0 in generation 18
Generation : 19
solution is found 2*x + y + 5 0.0 in generation 19
*****
The correct and final solution is : 2*x + y + 5
□
```

شکل 7 - خروجی کد به ازای تابع $2x + y + 5$



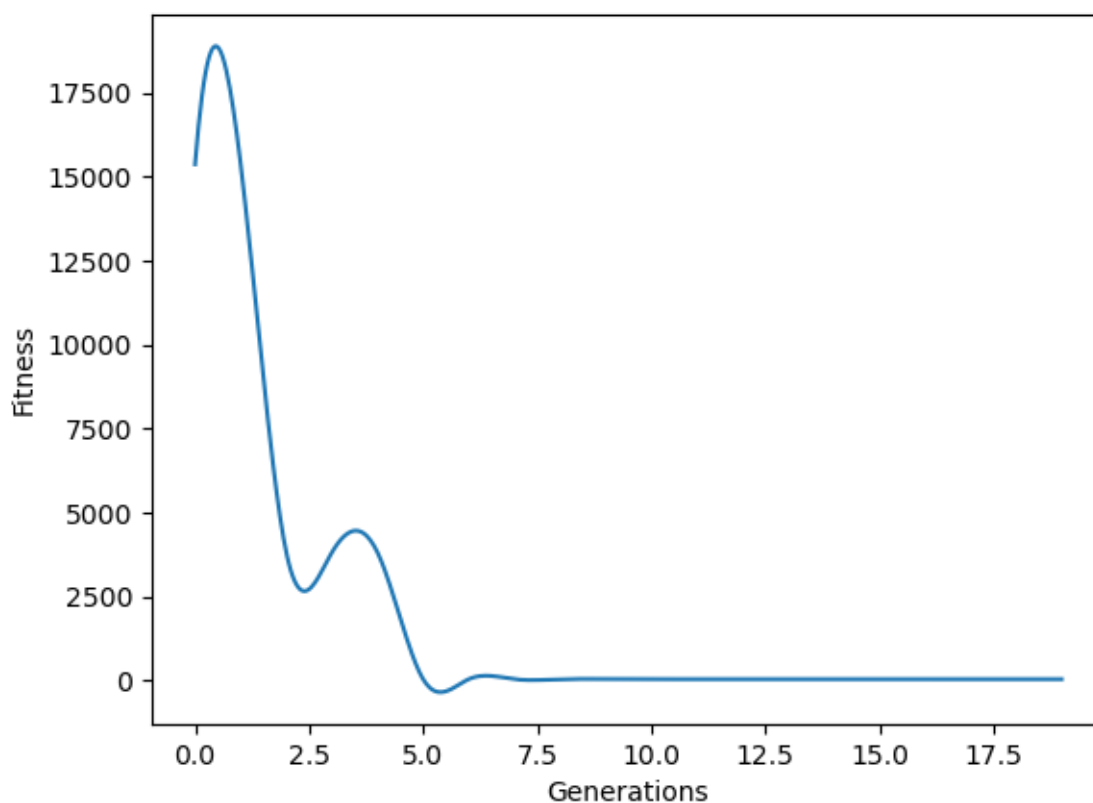
شکل 8 - نمودار fitness_function به ازای هر نسل

4. آزمایش چهارم : در این آزمایش تابع ما دارای 3 متغیر می باشد.

$$F(x,y,z) = x + 6y + z + 9$$

```
PS C:\Users\IT CITY\Documents\AI\Project> & "C:/Users/IT CITY/anaconda3/python.exe" "c:/Users/IT CITY/Documents/AI/Project/driver.py"
number of variables :3
enter function :x + 9 + 6 *y + z
Generation : 0
Generation : 1
Generation : 2
Generation : 3
Generation : 4
Generation : 5
Generation : 6
Generation : 7
Generation : 8
Generation : 9
Generation : 10
Generation : 11
Generation : 12
Generation : 13
Generation : 14
Generation : 15
solution is found x + 6*y + z + 9 0.0 in generation 15
Generation : 16
solution is found x + 6*y + z + 9 0.0 in generation 16
Generation : 17
solution is found x + 6*y + z + 9 0.0 in generation 17
Generation : 18
solution is found x + 6*y + z + 9 0.0 in generation 18
Generation : 19
solution is found x + 6*y + z + 9 0.0 in generation 19
PS C:\Users\IT CITY\Documents\AI\Project> []
```

شکل 9 - خروجی کد به ازای تابع $x + 6y + z + 9$



شکل 10 - نمودار fitness_function به ازای هر نسل