**по лабораторной работе № 8**

**Дисциплина: Оптимизация вычислительных процессов**

# Тема: Гибридный алгоритм оптимизации функций

Выполнил:

Студент 46/2 группы

Ролдугин А.С.

Преподаватель:

Климец А.А.

**Цель работы:** разработатьгибридный алгоритм оптимизации функций.

**1. Алгоритм бактериальной оптимизации функций.**

В работе за основу гибридных алгоритмов применим иммунный алгоритм AIS и алгоритм роя частиц PSO. AIS и PSO алгоритмы оптимизации обладают взаимодополняющими свойствами. Так, известно, что **алгоритмы AIS имеют высокую вероятностью локализации глобального экстремума целевой функции, но невысокую скоростью сходимости. Напротив, для алгоритмов PSO характерна высокая скорость сходимости**, но, если не прикладывать дополнительных усилий,

относительно невысокая вероятность локализации глобального экстремума. Гибридизация алгоритмов AIS и PSO преследует цель объединить указанные достоинства этих алгоритмов и нивелировать их недостатки. Реализуем в качестве подхода последовательную гибридизацию.

Последовательная гибридизация предполагает фиксированный порядок использования двух объединяемых алгоритмов, хотя момент времени переключения с одного алгоритма оптимизации на другой может не фиксироваться.

Схема гибридного алгоритма имеет следующий вид.

1) Инициализируем популяцию из S частиц, и выполняем фиксированное число итераций алгоритма PSO. Выбираем лучших полученных частиц.

2) Случайным образом создаем S2 = S1 антител- формируем популяцию S2.

3) Объединяем отобранные частицы и антитела в единую популяцию S размером S = S1 + S2 и вычисляем значения целевой функции для каждого из антител.

4) Клонируем n лучших антител пропорционально их аффинности – создаем временное множество клонов С.

5) Применяем мутацию к множеству С, образуя таким образом множество мутированных клонов С\*. 6) Выбираем лучшие элементы из множества клонов С\* для формирования множества клеток памяти М.

7) Обновляем множество антител S, записав вместо худших антител клетки памяти и сгенерированные для обеспечения разнообразия популяции антитела.

Проведем сравнение работы каждого алгоритма по отдельности (иммунный алгоритм и алгоритм роя частиц) и гибридного для функции Розенброка. Для каждого алгоритма сравним результаты на разных данных.

**2. Программная реализация алгоритма гибридного алгоритма:**

**def** Rosenbrock**(**x**,** y**):**

**return** **(**1 **-** x**)** **\*\*** 2 **+** 100 **\*** **(**y **-** x **\*\*** 2**)** **\*\*** 2

**def** Rastrigin**(**x**,** y**):**

**return** 20 **+** x **\*\*** 2 **+** y **\*\*** 2 **-** 10 **\*** **(**cos**(**2 **\*** pi **\*** x**)** **+** cos**(**2 **\*** pi **\*** y**))**

**def** Himmelblau**(**x**,** y**):**

**return** **(**x**\*\***2**+**y**-**11**)\*\***2 **+** **(**x**+**y**\*\***2**-**7**)\*\***2

**def** draw\_Rosenbrock**():**

**return** **(**1 **-** X**)** **\*\*** 2 **+** 100 **\*** **(**Y **-** X **\*\*** 2**)** **\*\*** 2

**def** draw\_Rastrigin**():**

**return** 20 **+** X **\*\*** 2 **+** Y **\*\*** 2 **-** 10 **\*** **(**np**.**cos**(**2 **\*** pi **\*** X**)** **+** np**.**cos**(**2 **\*** pi **\*** Y**))**

**def** draw\_Himmelblau**():**

**return** **(**X**\*\***2**+**Y**-**11**)\*\***2 **+** **(**X**+**Y**\*\***2**-**7**)\*\***2

**def** select\_func**(**name\_func**)** **:**

**if** name\_func **==** "Функция Розенброка" **:**

f **=** Rosenbrock

Z **=** draw\_Rosenbrock

**elif** name\_func **==** "Функция Растригина" **:**

f **=** Rastrigin

Z **=** draw\_Rastrigin

**else** **:**

f **=** Himmelblau

Z **=** draw\_Himmelblau

**return** f**,** Z

**class** **Particle(**object**):**

**def** \_\_init\_\_**(**self**,** swarm**):**

self**.**\_\_currentPosition **=** np**.**random**.**rand**(**

2**)** **\*** **(**swarm**.**maxvalues **-** swarm**.**minvalues**)** **+** swarm**.**minvalues

self**.**\_\_localBestPosition **=** self**.**\_\_currentPosition**[:]**

self**.**\_\_localBestFinalFunc **=** swarm**.**getFinalFunc**(**

self**.**\_\_currentPosition**)**

self**.**\_\_velocity **=** self**.**\_\_getInitVelocity**(**swarm**)**

*@property*

**def** position**(**self**):**

**return** self**.**\_\_currentPosition

*@property*

**def** velocity**(**self**):**

**return** self**.**\_\_velocity

**def** \_\_getInitVelocity**(**self**,** swarm**):**

minval **=** **-(**swarm**.**maxvalues **-** swarm**.**minvalues**)**

maxval **=** **(**swarm**.**maxvalues **-** swarm**.**minvalues**)**

**return** np**.**random**.**rand**(**2**)** **\*** **(**maxval **-** minval**)** **+** minval

**def** nextIteration**(**self**,** swarm**):**

rnd\_currentBestPosition **=** np**.**random**.**rand**(**swarm**.**dimension**)**

rnd\_globalBestPosition **=** np**.**random**.**rand**(**swarm**.**dimension**)**

veloRatio **=** swarm**.**localVelocityRatio **+** swarm**.**globalVelocityRatio

commonRatio **=** **(**2.0 **\*** swarm**.**currentVelocityRatio **/**

**(**np**.**abs**(**2.0 **-** veloRatio **-** np**.**sqrt**(**veloRatio **\*\*** 2 **-** 4.0 **\*** veloRatio**))))**

newVelocity\_part1 **=** commonRatio **\*** self**.**\_\_velocity

newVelocity\_part2 **=** **(**commonRatio **\***

swarm**.**localVelocityRatio **\***

rnd\_currentBestPosition **\***

**(**self**.**\_\_localBestPosition **-** self**.**\_\_currentPosition**))**

newVelocity\_part3 **=** **(**commonRatio **\***

swarm**.**globalVelocityRatio **\***

rnd\_globalBestPosition **\***

**(**swarm**.**globalBestPosition **-** self**.**\_\_currentPosition**))**

self**.**\_\_velocity **=** newVelocity\_part1 **+** newVelocity\_part2 **+** newVelocity\_part3

self**.**\_\_currentPosition **+=** self**.**\_\_velocity

finalFunc **=** swarm**.**getFinalFunc**(**self**.**\_\_currentPosition**)**

**if** finalFunc **<** self**.**\_\_localBestFinalFunc**:**

self**.**\_\_localBestPosition **=** self**.**\_\_currentPosition**[:]**

self**.**\_\_localBestFinalFunc **=** finalFunc

**class** **Swarm(**object**):**

**def** \_\_init\_\_**(**self**,** swarmsize**,** currentVelocityRatio**,** localVelocityRatio**,** globalVelocityRatio**):**

self**.**\_\_swarmsize **=** swarmsize

self**.**\_\_minvalues **=** np**.**array**([-**5**]** **\*** 2**)**

self**.**\_\_maxvalues **=** np**.**array**([**5**]** **\*** 2**)**

self**.**\_\_currentVelocityRatio **=** currentVelocityRatio

self**.**\_\_localVelocityRatio **=** localVelocityRatio

self**.**\_\_globalVelocityRatio **=** globalVelocityRatio

self**.**\_\_globalBestFinalFunc **=** **None**

self**.**\_\_globalBestPosition **=** **None**

self**.**\_\_swarm **=** self**.**\_\_createSwarm**()**

**def** \_\_getitem\_\_**(**self**,** index**):**

**return** self**.**\_\_swarm**[**index**]**

**def** \_\_createSwarm**(**self**):**

**return** **[**Particle**(**self**)** **for** \_ **in** range**(**self**.**\_\_swarmsize**)]**

**def** nextIteration**(**self**):**

**for** particle **in** self**.**\_\_swarm**:**

particle**.**nextIteration**(**self**)**

*@property*

**def** minvalues**(**self**):**

**return** self**.**\_\_minvalues

*@property*

**def** maxvalues**(**self**):**

**return** self**.**\_\_maxvalues

*@property*

**def** currentVelocityRatio**(**self**):**

**return** self**.**\_\_currentVelocityRatio

*@property*

**def** localVelocityRatio**(**self**):**

**return** self**.**\_\_localVelocityRatio

*@property*

**def** globalVelocityRatio**(**self**):**

**return** self**.**\_\_globalVelocityRatio

*@property*

**def** globalBestPosition**(**self**):**

**return** self**.**\_\_globalBestPosition

*@property*

**def** globalBestFinalFunc**(**self**):**

**return** self**.**\_\_globalBestFinalFunc

**def** getFinalFunc**(**self**,** position**):**

finalFunc **=** self**.**\_finalFunc**(**position**)**

**if** **(**self**.**\_\_globalBestFinalFunc **==** **None** **or** finalFunc **<** self**.**\_\_globalBestFinalFunc**):**

self**.**\_\_globalBestFinalFunc **=** finalFunc

self**.**\_\_globalBestPosition **=** position**[:]**

**return** finalFunc

**def** \_finalFunc**(**self**,** position**):**

x **=** position**[**0**]**

y **=** position**[**1**]**

f **=** select\_func**(**selected\_func**.**get**())[**0**]**

**return** f**(**x**,** y**)**

*@property*

**def** dimension**(**self**):**

**return** len**(**self**.**minvalues**)**

**class** **ImmuneCell:**

**def** \_\_init\_\_**(**self**,** x**,** y**):**

self**.**x **=** x

self**.**y **=** y

self**.**fitness **=** 0

self**.**f **=** select\_func**(**selected\_func**.**get**())[**0**]**

**def** randomPos**(**self**):**

self**.**x **=** uniform**(-**8**,** 8**)**

self**.**y **=** uniform**(-**8**,** 8**)**

**def** calcFitness**(**self**):**

self**.**fitness **=** 1 **/** self**.**f**(**self**.**x**,** self**.**y**)**

**def** mutate**(**self**):**

rate **=** 2**\*\*(-**float**(**mutationEffect**.**get**())** **\*** self**.**fitness**)**

**if** rate **<** 0.0001**:**

rate **=** 0.0001

self**.**x **+=** uniform**(-**rate **\*** float**(**area**.**get**()),** rate **\*** float**(**area**.**get**()))**

self**.**y **+=** uniform**(-**rate **\*** float**(**area**.**get**()),** rate **\*** float**(**area**.**get**()))**

**def** clone**(**self**):**

clone **=** ImmuneCell**(**self**.**x**,** self**.**y**)**

# clone.x = self.x

# clone.y = self.y

clone**.**fitness **=** self**.**fitness

**return** clone

**def** rotate**():**

plt**.**show**()**

**def** calc**(**populationSize**,** numOfBest**,** fertility**,** mutationEffect**,** area**,** maxIterImun**,** maxIterSwarm**,** swarmsize**)** **:**

currentVelocityRatio **=** 0.1

localVelocityRatio **=** 1.0

globalVelocityRatio **=** 5.0

start\_time **=** time**.**time**()**

swarm **=** Swarm**(**swarmsize**,** currentVelocityRatio**,**

localVelocityRatio**,** globalVelocityRatio**)**

**for** n **in** range**(**maxIterSwarm**):**

swarm**.**nextIteration**()**

best **=** **[**swarm**.**globalBestPosition**[**0**],** swarm**.**globalBestPosition**[**1**]]**

population **=** **[]**

**for** i **in** range**(**populationSize**):**

cell **=** ImmuneCell**(**best**[**0**],** best**[**1**])**

cell**.**calcFitness**()**

population**.**append**(**cell**)**

c **=** 0

**while** c **<** maxIterImun**:**

population **=** sorted**(**population**,** key**=lambda** x**:** x**.**fitness**,** reverse**=True)[**

**:**numOfBest**]**

numberOfClones **=** populationSize **-** 2 **\*** numOfBest

**for** i **in** range**(**numOfBest**):**

clones **=** **[]**

numberOfClones **=** int**(**numberOfClones **/** 2**)** **+** 1

**for** j **in** range**(**numberOfClones **\*** fertility**):**

clone **=** population**[**i**].**clone**()**

clone**.**mutate**()**

clone**.**calcFitness**()**

clones**.**append**(**clone**)**

clones **=** sorted**(**clones**,** key**=lambda** x**:** x**.**fitness**,** reverse**=True)**

**for** i **in** range**(**4**):**

population**.**append**(**clones**[**i**])**

cell **=** ImmuneCell**(**best**[**0**],** best**[**1**])**

cell**.**randomPos**()**

cell**.**calcFitness**()**

population**.**append**(**cell**)**

cell **=** ImmuneCell**(**best**[**0**],** best**[**1**])**

cell**.**randomPos**()**

cell**.**calcFitness**()**

population**.**append**(**cell**)**

**if** clones**[**0**].**fitness **>** population**[**i**].**fitness**:**

population**[**i**]** **=** clones**[**0**]**

population **=** sorted**(**population**,** key**=lambda** x**:** x**.**fitness**,** reverse**=True)[**

**:**numOfBest**]**

c **+=** 1

**print(**"Лучшая частица:"**,** best**)**

**print(**"Результат - лучшая иммунная клетка:"**,** population**[**0**].**x**,**

population**[**0**].**y**,** 1 **/** population**[**0**].**fitness**)**

**print(**"Время: "**,** time**.**time**()** **-** start\_time**)**

draw**(**population**[**0**])**

**def** draw**(**population**)** **:**

ax**.**scatter**(**population**.**x**,** population**.**y**,** 1 **/** population**.**fitness**,** color**=**'red'**,** s**=**50**,** marker**=**'o'**)**

lbl **=** "Найденный минимум\nx: " **+** str**(**population**.**x**)** **+** "\ny: " **+** str**(**population**.**y**)** **+** "\nz: " **+** str**(**1 **/** population**.**fitness**)**

tk**.**Label**(**root**,**text**=**lbl**).**grid**(**row**=**19**,** column**=**20**,** columnspan **=** 2**)**

**def** draw\_graphic**(**name\_func**)** **:**

Z **=** select\_func**(**name\_func**)[**1**]**

plt**.**cla**()**

ax**.**plot\_surface**(**X**,** Y**,** Z**(),** color **=** "green"**)**

canvas**.**draw**()**

root **=** tk**.**Tk**()**

fig **=** plt**.**figure**(**dpi**=**100**)**

ax **=** fig**.**add\_subplot**(**111**,** projection**=**'3d'**)**

canvas **=** FigureCanvasTkAgg**(**fig**,** master**=**root**)**

canvas**.**mpl\_connect**(**

"key\_press\_event"**,** **lambda** event**:** **print(**f"you pressed {event.key}"**))**

canvas**.**mpl\_connect**(**"key\_press\_event"**,** key\_press\_handler**)**

canvas**.**get\_tk\_widget**().**grid**(**row **=** 0**,** column **=** 0**,**rowspan**=**20**,** columnspan **=** 20**)**

toolbar **=** NavigationToolbar2Tk**(**canvas**,** root**,** pack\_toolbar**=False)**

toolbar**.**update**()**

toolbar**.**grid**(**row**=**21**,** column**=**0**,** columnspan **=** 20**,** rowspan **=** 1**)**

selected\_func **=** StringVar**(**value**=**"Функция Розенброка"**)**

funcs **=** **[**"Функция Розенброка"**,** "Функция Растригина"**,** "Функция Химмельблау"**]**

func\_list\_box **=** tk**.**OptionMenu**(**root**,** selected\_func**,** **\***funcs**,** command **=** draw\_graphic **)**

func\_list\_box**.**config**(**font **=** "times 12"**,** width **=** 20**)**

func\_list\_box**.**grid**(**row **=** 1**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Размер популяции (иммунный):"**).**grid**(**row **=** 2**,** column**=**20**,** columnspan **=** 2**)**

populationSize **=** tk**.**Entry**(**root**)**

populationSize**.**grid**(**row **=** 3**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Число приоритетных особей:"**).**grid**(**row **=** 4**,** column**=**20**,** columnspan **=** 2**)**

numOfBest **=** tk**.**Entry**(**root**)**

numOfBest**.**grid**(**row **=** 5**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Коэффициент рождаемости:"**).**grid**(**row **=** 6**,** column**=**20**,** columnspan **=** 2**)**

fertility **=** tk**.**Entry**(**root**)**

fertility**.**grid**(**row **=** 7**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Интенсивность мутации:"**).**grid**(**row **=** 8**,** column**=**20**,** columnspan **=** 2**)**

mutationEffect **=** tk**.**Entry**(**root**)**

mutationEffect**.**grid**(**row **=** 9**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Область мутации:"**).**grid**(**row **=** 10**,** column**=**20**,** columnspan **=** 2**)**

area **=** tk**.**Entry**(**root**)**

area**.**grid**(**row **=** 11**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Кол-во итераций (иммунный):"**).**grid**(**row **=** 12**,** column**=**20**,** columnspan **=** 2**)**

maxIter **=** tk**.**Entry**(**root**)**

maxIter**.**grid**(**row **=** 13**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Кол-во итераций (рой):"**).**grid**(**row **=** 14**,** column**=**20**,** columnspan **=** 2**)**

maxIterSwarm **=** tk**.**Entry**(**root**)**

maxIterSwarm**.**grid**(**row **=** 15**,** column **=** 20**,** columnspan **=** 2**)**

tk**.**Label**(**root**,** text**=**"Размер популяции (рой):"**).**grid**(**row **=** 16**,** column**=**20**,** columnspan **=** 2**)**

swarmsize **=** tk**.**Entry**(**root**)**

swarmsize**.**grid**(**row **=** 17**,** column **=** 20**,** columnspan **=** 2**)**

rotateBtn **=** tk**.**Button**(**master**=**root**,** text**=**"Обзор 3D"**,** command**=**rotate**,** width **=** 10**)**

rotateBtn**.**grid**(**row **=** 21**,** column **=** 20**,** columnspan **=** 1**)**

rotateBtn **=** tk**.**Button**(**master**=**root**,** text**=**"Обзор 3D"**,** command**=**rotate**,** width **=** 10**)**

rotateBtn**.**grid**(**row **=** 21**,** column **=** 20**,** columnspan **=** 1**)**

calcBtn **=** tk**.**Button**(**master**=**root**,** text **=** "Старт"**,** width **=** 10**,** command **=**

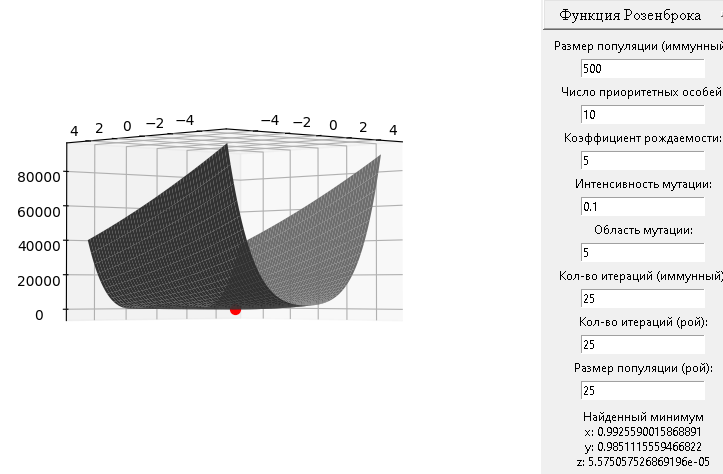
**lambda:** calc**(**int**(**populationSize**.**get**()),** int**(**numOfBest**.**get**()),** int**(**fertility**.**get**()),**

float**(**mutationEffect**.**get**()),** float**(**area**.**get**()),** int**(**maxIter**.**get**()),** int**(**maxIterSwarm**.**get**()),** int**(**swarmsize**.**get**())))**

calcBtn**.**grid**(**row**=**21**,** column **=** 21**,** columnspan **=** 1**)**

root**.**mainloop**()**

Рисунок 1 – Программная реализация гибридного алгоритма



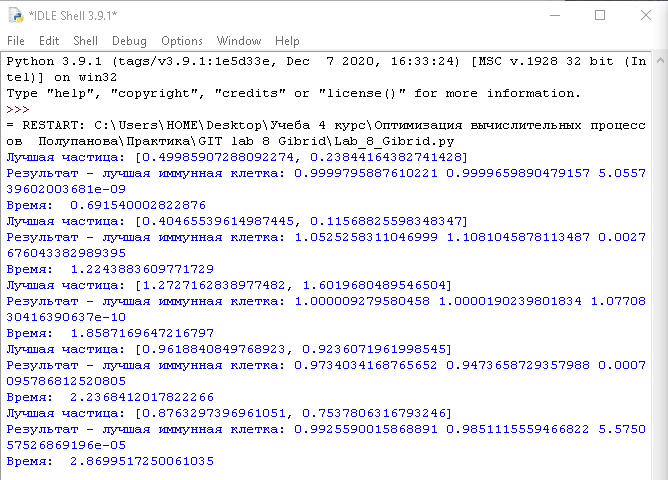
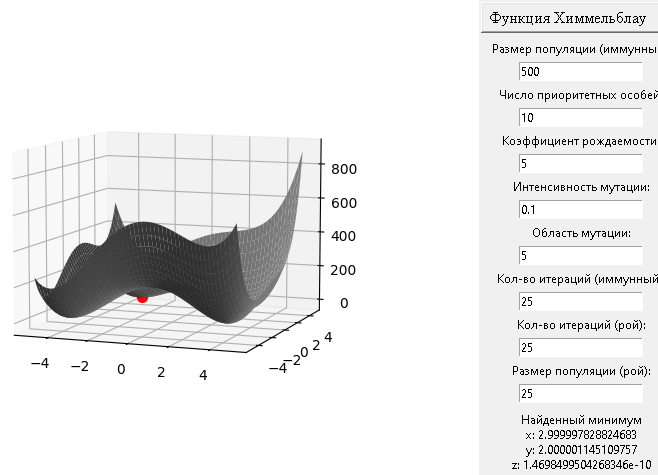


Рисунок 3 – Результат работы алгоритма – функция Розенброка



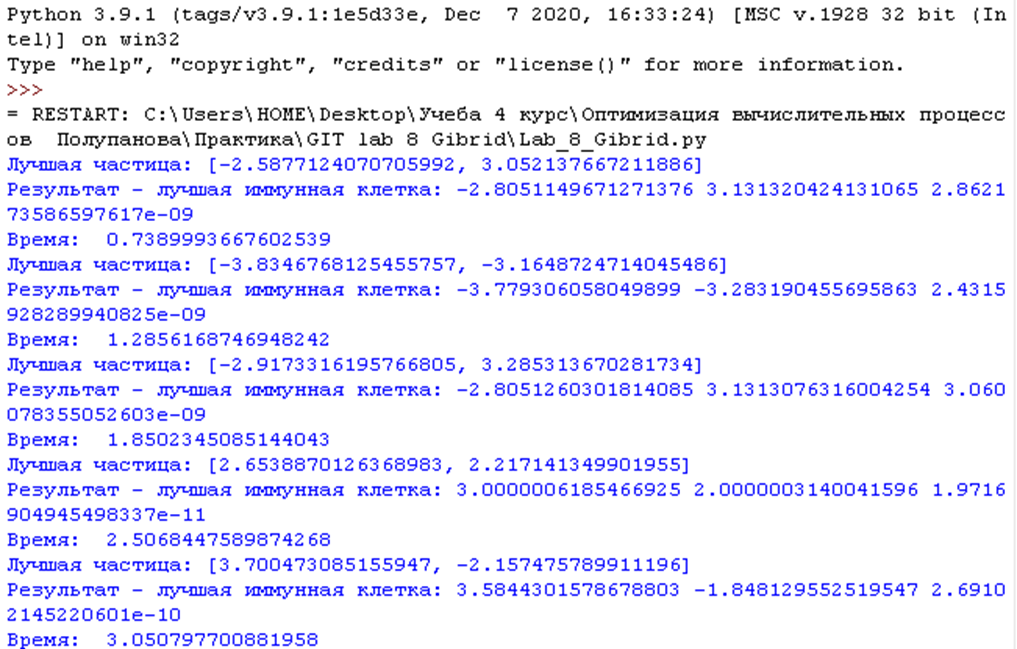
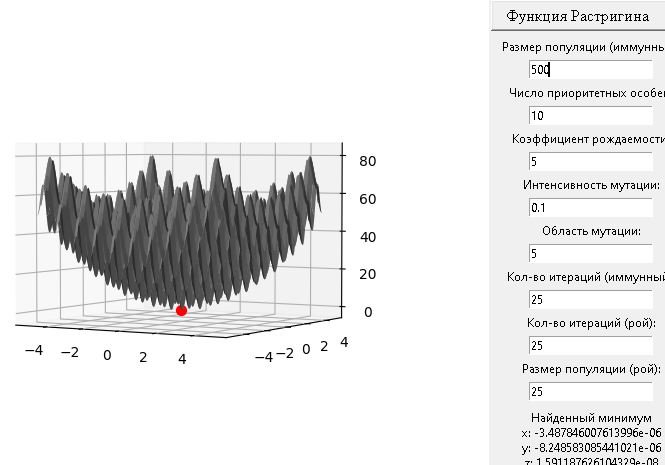


Рисунок 4 – Результат работы алгоритма – функция Химмельблау



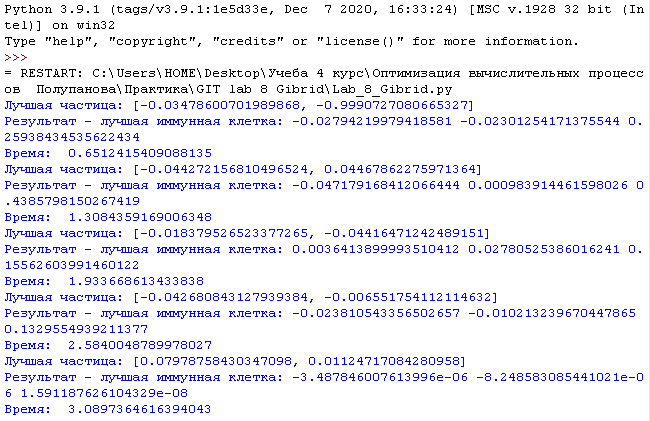


Рисунок 5 – Результат работы алгоритма – функция Растригина

Рисунок 6 – График сравнения работы алгоритмов

Из данных на Рисунке и значений времени работы алгоритмов оптимизации функции Розенброка для каждого из алгоритмов можно сделать вывод, что при приблизительно равных значениях функции, гибридный алгоритм является наиболее быстрым, при этом иммунный алгоритм оптимизации показал самый медленный результат.

**Вывод:**

В ходе лабораторной работы был реализован гибридный алгоритм. Алгоритм сочетает в себе лучшие свойства базовых алгоритмов - быстрая сходимость алгоритма роя частиц PSO и высокая вероятность локализации глобального экстремума алгоритма искусственной иммунной сети AIS.