Лабораторная работа

Тема: генетические алгоритмы

Цель работы: получение практических навыков использования генетических алгоритмов на языке Python с использованием библиотеки DEAP.

Задание: используя программу Jupiter Notebook, язык программирования Python, библиотеку DEAP, NumPy, Matplotlib и др. реализовать генетический алгоритм согласно варианту.

Отчёт по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание, вариант.
2. Описание построенного генетического алгоритма и его операторов.
3. Протокол прогона ГА: особи популяций, лучшие особи, приспособленность особей, максимальное значение приспособленности, минимальное значение приспособленности.
4. График изменения параметров ГА.
5. Выполнить 3 прогона с разными параметрами генетического алгоритма, сравнить результаты и определить лучший вариант параметров (который быстрее привёл к результату / дал лучший результат).
6. Код.

Варианты

	Целевая функция	Вид особи	Приспособленность	Отбор	Кроссовер	Мутация	Стратегия
1	Максимум, однокритериальная	Тип: массив вещественных чисел (array), значения: от 0 до 1 могут повторятся, длина особи: 10	Среднее значение генов особей популяции делённая на 5 ген особи	Турнирный отбор	Одноточечный	Полиномиальная мутация	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
2	Минимум, однокритериальная	Тип: список целых чисел, значения: от 0 до 10 не могут повторятся, длина особи: 5	Сумма значений генов особи за исключением чётных генов	Отбор рулеткой	Упорядоченный	Меняет на противоположное значение ген	(μ+λ)
3	Максимум, однокритериальная	Тип: массив вещественных чисел (numpy), значения: от 0 до 13 могут повторятся, длина особи: 10	Среднее значение генов особей популяции	Выбор случайных k особей	Двухточечный	Гауссовская мутация	(μ, λ)
4	Минимум, однокритериальная	Тип: первый параметр булевый, длина особи: 7	Количество элементов =1 (истина)	Выбор лучших k особей	Равномерный	Меняет на противоположное значение ген	(μ+λ)
5	Минимум, однокритериальная	Тип: массив вещественных чисел (array), значения: от 0 до 1 могут повторятся, длина особи: 10	Среднее значение генов особей популяции	Выбор лучших k особей	Равномерный	Гауссовская мутация	(μ+λ)
6	Максимум, однокритериальная	Тип: список целых чисел, значения: от 0 до 10 не могут повторятся, длина особи: 5	Сумма значений генов особи за исключением нечётных генов	Выбор случайных k особей	Упорядоченный	Меняет на противоположное значение ген	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
7	Минимум, однокритериальная	Тип: массив вещественных чисел (numpy), значения: от 0 до 13 могут повторятся, длина особи: 10	Среднее значение генов особей популяции делённая на 2 ген особи	Отбор рулеткой	Двухточечный	Полиномиальная мутация	(μ, λ)
8	Максимум, однокритериальная	Тип: целочисленный в интервале от 0 до 5 могут повторяться, длина особи: 17	сумма значении генов	Турнирный отбор	Равномерный	Меняет на противоположное значение ген	(μ+λ)
9	Максимум, однокритериальная	Тип: массив вещественных чисел (array), значения: от 0 до 1 могут повторятся, длина особи: 10	Среднее значение генов особей популяции делённая на 5 ген особи	Турнирный отбор	Двухточечный	Полиномиальная мутация	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
10	Минимум, однокритериальная	Тип: список целых чисел, значения: от 0 до 10 не могут повторятся, длина особи: 5	Сумма значений генов особи за исключением чётных генов	Выбор лучших k особей	Упорядоченный	Меняет на противоположное значение ген	(μ+λ)
11	Максимум, однокритериальная	Тип: массив вещественных чисел (numpy), значения: от 0 до 13 могут повторятся, длина особи: 10	Среднее значение генов особей популяции	Выбор случайных k особей	Равномерный	Гауссовская мутация	(μ, λ)

12	Минимум, однокритериальная	Тип: булевый длина особи: 10	Количество элементов =1 (истина) минус 5 ген	Выбор случайных k особей	Одноточечный	Полиномиальная мутация	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
13	Минимум, однокритериальная	Тип: массив вещественных чисел (array), значения: от 0 до 1 могут повторятся, длина особи: 10	Среднее значение генов особей популяции	Турнирный отбор	Одноточечный	Гауссовская мутация	(μ+λ)
14	Максимум, однокритериальная	Тип: список целых чисел, значения: от 0 до 10 не могут повторятся, длина особи: 5	Сумма значений генов особи за исключением нечётных генов	Турнирный отбор	Упорядоченный	Меняет на противоположное значение ген	(μ+λ)
15	Минимум, однокритериальная	Тип: массив вещественных чисел (numpy), значения: от 0 до 13 могут повторятся, длина особи: 10	Среднее значение генов особей популяции делённая на 2 ген особи	Турнирный отбор	Равномерный	Гауссовская мутация	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
16	Максимум, однокритериальная	Тип: целочисленный в интервале от 0 до 5 длина особи: 7	1) Количество элементов =1 (истина)	Турнирный отбор	Одноточечный	Меняет на противоположное значение ген	(μ+λ)
17	Максимум, однокритериальная	Тип: массив вещественных чисел (array), значения: от 0 до 1 могут повторятся, длина особи: 10	Среднее значение генов особей популяции делённая на 5 ген особи	Выбор случайных k особей	Двухточечный	Гауссовская мутация	(μ, λ)
18	Минимум, однокритериальная	Тип: список целых чисел, значения: от 0 до 10 не могут повторятся, длина особи: 5	Сумма значений генов особи за исключением чётных генов	Выбор лучших k особей	Упорядоченный	Меняет на противоположное значение ген	(μ+λ)
19	Максимум, однокритериальная	Тип: массив вещественных чисел (numpy), значения: от 0 до 13 могут повторятся, длина особи: 10	Среднее значение генов особей популяции	Турнирный отбор	Одноточечный	Гауссовская мутация	(μ, λ)
20	Минимум, однокритериальная	Тип: вещественный в интервале от 0 до, значения не повторяются, длина особи: 14	среднее значение ген	Отбор рулеткой	Двухточечный	Полиномиальная мутация	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
21	Минимум, однокритериальная	Тип: массив вещественных чисел (array), значения: от 0 до 1 могут повторятся, длина особи: 10	Среднее значение генов особей популяции	Отбор рулеткой	Одноточечный	Гауссовская мутация	(μ+λ)
22	Максимум, однокритериальная	Тип: список целых чисел, значения: от 0 до 10 не могут повторятся, длина особи: 5	Сумма значений генов особи за исключением нечётных генов	Выбор лучших k особей	Равномерный	Меняет на противоположное значение ген	(μ, λ)
23	Минимум, однокритериальная	Тип: массив вещественных чисел (numpy), значения: от 0 до 13 могут повторятся, длина особи: 10	Среднее значение генов особей популяции делённая на 2 ген особи	Выбор случайных k особей	Одноточечный	Гауссовская мутация	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.

Методические указания по использованию библиотеки DEAP для работы с генетическими алгоритмами

Элементы библиотеки DEAP для генетического алгоритма

Библиотека DEAP (Distributed Evolutionary Algorithms in Python) содержит распределённые эволюционные алгоритмы в Python.

Для реализации этих эволюционных алгоритмов, требуется использовать пакеты библиотеки, среди которых можно выделить ядро, реализующее базовые классы и инструменты для алгоритмов, и пакеты, реализующие различные эволюционные методы (Рисунок 1, Таблица 1).

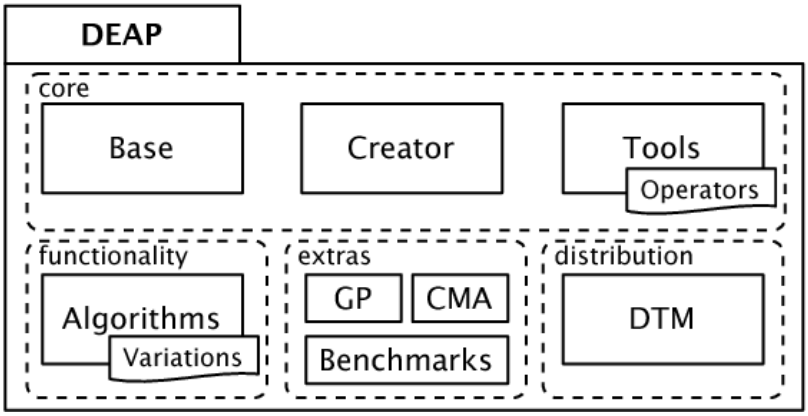

Рисунок 1 – Схема классов библиотеки[1]

Таблица 1 - Модули библиотеки для реализации ГА

Модуль	Описание
deap.creator	Содержит класс `deap.creator.create`, `необходим для создания новых классов` с именем name, наследующий классы из пакета base. Новый класс может иметь аразные трибуты.
deap.base	Пакет содержит базовые классы. 1) deap.base.Toolbox - набор инструментов для эволюционных методов (на основе классов пакета tools), 2) `deap.base.Fitness` - пригодность / приспособленность (показатель качества решения) особи. 3) *class* `deap.base.Tree (для версии DEAP 0.8.2)` Базовый класс N-арного дерева. Дерево инициализируется из содержимого списка. Первый элемент списка является корнем дерева, затем следующие элементы являются узлами. Каждый узел может быть либо списком, либо отдельным элементом. В случае списка это рассматривается как поддерево, иначе лист.
deap.tools	Модуль инструментов содержит операторы для эволюционных алгоритмов. Набор операторов, которые он содержит, доступен на панели инструментов, также модуль содержит служебные инструменты для фиксации данных о функционировании алгоритмов.
deap.algorithms	Модуль алгоритмов содержит основные реалиации эволюционных методов, при этом используются операторы, зарегистрированные в соответсвующем объекте Toolbox. Обычно используются следующие ключевые слова: mate () для кроссовера, mutate () для мутации, select ()

[1] https://www.semanticscholar.org/paper/DEAP%3A-a-python-framework-for-evolutionary-Rainville-Fortin/7627ef0d9975dc50f81f6894f976336c31bb6a38

	для выбора.

Реализация генетического алгоритма

Определение приспособленности особей и вида особи (индивида)

Необходимо определить класс для приспособленности и особи (индивида), для этого нужно использовать пакет deap.creator и класс Creator:

`deap.creator.create`(*name*, *base*[, *attribute*[, ...]])

Первый параметр *name* задаёт имя класса. Базой (родителем) для пользовательского класса приспособленности является base.Fitness (можно определить другой класс, если он реализован программистом).

При использовании алгоритмов deap в качестве третьего атрибута необходимо задать параметр *weights*. Нужно определить тип задачи оптимизации (максимум / минимум и однокритериальная / многокритериальная). В зависимости от этого определяется параметр weights (Таблица 2). Веса могут также использоваться, чтобы варьировать важность критериев оптимизации друг относительно друга (чем больше вес, тем важнее критерий), т.е. для задание многоцелевой функции оптимизации. Это означает, что весами могут быть любые действительные числа, и только знак используется для определения того, выполняется ли максимизация или минимизация.

Таблица 2 – Примеры определения параметра weights

Тип задачи оптимизации	Значения параметра weights
Однокритериальная на минимум	weights=(-1.0,)
Однокритериальная на максимум	weights=(1.0,)
Многокритериальная оптимизация (2 критерия)	weights=(-1.0, 1.0)

Пример:

1) Однокритериальная оптимизация на максимум

```
from deap import creator
creator.create("F", base.Fitness, weights=(1.0,))
```

2) Однокритериальная минимизация с именем FitnessMin.

```
from deap import creator
creator.create("FitnessMin", base.Fitness, weights = (- 1.0,))
```

3) 3) Этот код создаёт соответствие, которое минимизирует первую цель и максимизирует вторую.

```
from deap import creator
creator.create ("FitnessMulti", base.Fitness, weights = (- 1.0, 1.0))
```

Для определения вида особи используется этот же класс, созданный ранее класс приспособленности становится значением параметра *fitness*.

В качестве особи может использоваться массив, причём его элементы могут быть разного типа. Для определения типа массива можно использовать параметр *typecode* (Таблица 3).

Таблица 3 – Коды типов

Код типа	Тип в python	Минимальный размер в байтах
'b'	int	1
'B'	int	1

'h'	int	2
'H'	int	2
'i'	int	2
'I'	int	2
'l'	int	4
'L'	int	4
'q'	int	8
'Q'	int	8
'f'	float	4
'd'	float	8

Пример:

1) Особь - простой список, содержащий вещественные числа

```
creator.create("Individual_1", list, fitness=creator.F)
```

2) Особь – массив из пакета array

```
import array
creator.create("Individual_2", array.array, typecode='b',
fitness=creator.FitnessMin)
```

3) Особь – массив из пакета numpy

```
import numpy
creator.create("Individual_3", numpy.ndarray, fitness=creator.FitnessMulti)
```

Регистрация операторов генетического алгоритма

Для определения параметров и операторов ГА используется класс base.Toolbox(), создаётся объект этого класса с помощью метода base.Toolbox(). В нем регистрируются операторы ГА:

`register`(*alias*, *method*[, *argument*[, ...]]).

Методу даётся псевдоним и указываются аргументы. Основные операторы ГА это:

- Инициализация (генерация особи)
- Формирование популяции
- Отбор родителей
- Скрещивание или кроссовер
- Мутация
- Миграция (для островной модели)

Для каждого оператора существуют разные методы и в модуле представлены варианты их реализаций (Таблица 4), кроме того программист может определить свои методы и задать их в качестве параметра функции *`register`*.

Таблица 4 – Список методов для реализации операторов ГА

Инициали-зация	Отбор	Скрещивание	Мутация
initRepeat()	selTournament()	cxOnePoint()	mutGaussian()
initIterate()	selRoulette()	cxTwoPoint()	mutShuffleIndexes()
initCycle()	selNSGA2()	cxUniform()	mutFlipBit()
	selNSGA3()	cxPartialyMatched()	mutPolynomialBounded()
	selSPEA2()	cxUniformPartialyMatched()	mutUniformInt()
	selRandom()	cxOrdered()	mutESLogNormal()
	selBest()	cxBlend()	
	selWorst()	cxESBlend()	

	selTournamentDCD()	cxESTwoPoint()	
	selDoubleTournament()	cxSimulatedBinary()	
	selStochasticUniversalSampling()	cxSimulatedBinaryBounded()	
	selLexicase()	cxMessyOnePoint()	
	selEpsilonLexicase()		**Миграция**
	selAutomaticEpsilonLexicase()		migRing()

Инициализация

Если значения генов выбираются случайно и могут повторяться, то можно использовать *initRepeat()*, т.е. случайная перестановка допустимых значений генов.

Если необходимо создать особь, значение генов в которой не повторяются, то можно использовать *initIterate()*.

Если необходимо генерировать хромосомы с заданной структурой (например 2 числа, одно целое другое вещественное, или 2 символа: первый цифра, второй латинская буква, третий – русская) и известно сколько хромосом должно быть в особи (сколько паз повторяться сочетания), то можно использовать функцию *initCycle()*.

Пример:

1) Инициализация значениями 0 или 1, причём длина особи = 10.

```
import random
from deap import tools
…
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual_1,
toolbox.attr_bool, 10)
```

2) Особи из 5 ген, которые принимают значения от 0 до 10.

```
import random
from deap import tools
…
toolbox.register("indices", random.sample, range(10), 5)
toolbox.register("individual", tools.initIterate, creator.Individual_1,
toolbox.indices)
```

3) Циклическая инициализация в особи 4 пары чисел, в паре первое число целое в промежутке от 5 до 10, второе вещественное, в интервала от 0.1 до 0.7

```
import random
from deap import tools
…
toolbox = base.Toolbox()
toolbox.register("attr_int", random.randint, 5, 10)
toolbox.register("attr_flt", random.uniform, 0.1, 0.7)
toolbox.register("individual", tools.initCycle, creator.Individual_1,
(toolbox.attr_int, toolbox.attr_flt), n=4)
```

Отбор

Существует много стратегий отбора, в модуле tools реализовано 14 (Таблица 4), рассмотрим 5 из них (Таблица 5), для которых нужен следующий набор параметров:

- individuals – особи;
- k – количество отбираемых особей;
- fit_attr – атрибут, по которому осуществляется отбор;
- tournsize - количество участников тура (для турнирного отбора).

Таблица 5 – Пример операторов отбора для ГА

Функция	**Параметры**	**Вид**	**Описание**
`deap.tools.selTournament`	(*individuals, k, tournsize, fit_attr='fitness'*)	`Турнирный отбор`	из популяции, содержащей m особей, выбирается случайным образом t особей и выбирается наиболее приспособленная (между выбранными особями проводится турнир), эта операция повторяется m раз
deap.tools.selRoulette	*(individuals, k, fit_attr='fitness')*	`Отбор рулеткой`	вид пропорционального отбора, когда особи отбираются с помощью n «запусков» рулетки (колесо рулетки содержит по одному сектору для каждого члена популяции, размер i-ого сектора пропорционален соответствующей величине $P_s(i)$)
`deap.tools.selRandom`	*(individuals, k)*	`Выбор случайных k особей`	`Случайная отбор`
`deap.tools.selBest`	(*individuals, k, fit_attr='fitness'*)	`Выбор лучших k особей`	`Отбор усечением`
`deap.tools.selWorst`	(*individuals, k, fit_attr='fitness'*)	`Выбор худших k особей`	`Отбор усечением наоборот`

Подробнее см. https://deap.readthedocs.io/en/master/api/tools.html.

Пример:

```
toolbox.register("select", tools.selTournament, tournsize=3)
```

Скрещивание

В модуле tools представлена реализация 12 видов операторов кроссовера (Таблица 4). Рассмотрим подробнее 4 из них (

Таблица 6), для них требуются переменные:

- ind1 – первый родитель
- ind2 – второй родитель
- indpb – вероятность (для равномерного кроссовера)

Таблица 6 – Пример операторов кроссовера

Функция	**Параметры**	**Тип кроссовера**	**Описание**	**Пример**
cxOnePoint()	(*ind1*, *ind2*)	Одноточечный	выбирается одна точка разрыва и родительские хромосомы обмениваются одной из получившихся частей	**Родитель 1:** 1001011|01001 **Родитель 2:** 0100011|00111 **Потомок 1:** 1001011|00111 **Потомок 2:** 0100011|01001
cxTwoPoint()	(*ind1*, *ind2*)	Двухточечный	выбираются две точки разрыва и родительские хромосомы обмениваются сегментом, который находится между двумя	**Родитель 1:** 100|101101|001 **Родитель 2:** 010|001100|111 **Потомок 1:** 100|001100|001 **Потомок 2:** 010|101101|111

			этими точками	
`deap.tools.cxUniform`	(*ind1*, *ind2*, *indpb*)	Равномерный	каждый бит первого потомка случайным образом наследуется от одного из родителей, второму потомку достается бит другого родителя	**Родитель 1:** 100101101001 **Родитель 2:** 010001100111 Вероятность: 90 % Случайные числа (100): 2, 24, 8, 93, 55, 13, 67, 43, 99, 61, 5, 89 **Потомок 1:** 100001100001 **Потомок 2:** 010101101111
`deap.tools.cxOrdered`	(*ind1*, *ind2*)	Упорядоченный	1) Выбор двух точек разрыва. 2) Обмен центральными частями 2) Обход всех незадействованных генов в особи, начиная со второй точки разрыва (гены значения которых не в середине, остаются на месте, повторяющиеся (были в середине), заменяются следующим неповторяющимся значением из этой же особи).	**Родитель 1:** 123|4567|89 **Родитель 2:** 375|2814|96 **Потомок 1:** 567 281493 **Потомок 2:** 281456793 9 – остался на месте, остальные сдвинулись

Подробнее см. https://deap.readthedocs.io/en/master/api/tools.html.

Пример:

```
toolbox.register("mate", tools.cxTwoPoint)
```

Мутация

Существует много стратегий отбора, в модуле tools реализовано 6 (Таблица 4), рассмотрим 3 из них (Таблица 7), для которых нужен следующий набор параметров:

- individual – особь,
- mu – среднее или последовательность средних для гауссовой аддитивной мутации,
- sigma – стандартное отклонение или последовательность стандартных отклонений для гауссовой аддитивной мутации,
- indpb – вероятность мутации,
- eta – степень скопления мутаций: высокая создаст мутанта, похожего на своего родителя, маленькая эта даст больше отличий,
- low – значение или последовательность значений, являющаяся нижней границей пространства поиска,
- up – значение или последовательность значений, являющаяся верхней границей пространства поиска.

Таблица 7 – Пример операторов мутации

Функция	**Параметры**	**Описание**
deap.tools.mutGaussian	(*individual*, *mu*, *sigma*, *indpb*)¶	добавляет случайное число,

		заданное гауссовым распределением с нулевым средним для каждого компонента входного вектора (применяется для вещественного типа)
deap.tools.mutFlipBit	(*individual*, *indpb*)	Меняет на противоположное значение бит с определённой вероятностью (применяется к логическому типу)
deap.tools.mutPolynomialBounded	(*individual*, *eta*, *low*, *up*, *indpb*)	Полиномиальная мутация

Подробнее см. https://deap.readthedocs.io/en/master/api/tools.html.

Пример:

1) Гауссовская мутация

```
toolbox.register("attr", random.random)
…
toolbox.register("mutate", tools.mutGaussian, mu=0.0, sigma=0.2, indpb=0.2)
```

2) Инверсия бит

```
toolbox.register("attr", random.randint, 0, 1)
…
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
```

3) Полиномиальная мутация

```
toolbox.register("attr", random.random)
…
toolbox.register("mutate", tools.mutPolynomialBounded, eta=10.0, low=0.1,
up=1, indpb=0.4)
```

Определение популяции

Популяция может быть представлена в виде массива особей, матрицы, роя (алгоритма роя), подпопуляций (для островной модели). Необходимо определить имя в наборе инструментов Toolbox, тип для популяции (список или массив с определённой размерностью), объект, который описывает особи (индивиды), и функцию генерации популяции (это может быть например tools.initRepeat как и для генерации особи).

Пример:

```
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

Определение инструментов контроля функционирования генетического алгоритма

Для фиксации эволюционных изменений в популяциях можно использовать готовые глассы из модуля tools (Таблица 8).

Таблица 8 – Пример классов для хранения данных о функционировании ГА

Класс		Описание
`deap.tools.Statistics`	([*key*]) – необязательный параметр, идентификатор для доступа к сохраняемым значениям, значение, возвращаемое ключом, может	собирает статистику по списку произвольных объектов, объект хранения регистрируется с помощью метода *register*

	быть многомерным объектом	
`deap.tools.Logbook`	нет	Объект - эволюционные записи в виде хронологического списка словарей. Данные могут быть получены с помощью метода select.
`deap.tools.HallOfFame`	(*maxsize*, *similar=<built-in function eq>*) *Maxsize* – максимальное количество особей в зале славы *similar* - Оператор эквивалентности между двумя особями (необязательный параметр)	Зал славы содержит лучшую особи популяции в процессе эволюции, особи лексикографически отсортированы, первый элемент Зала славы особь с максимальной приспособленностью.

Подробнее см. https://deap.readthedocs.io/en/master/api/tools.html.

Пример:

1) Статистика и зал славы

```
hof = tools.HallOfFame(1)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", numpy.mean)
stats.register("std", numpy.std)
stats.register("min", numpy.min)
stats.register("max", numpy.max)
```

2) Книга логов

```
log = Logbook()
log.record(gen=0, mean=5.4, max=10.0)
log.record(gen=1, mean=9.4, max=15.0)
log.select("mean")
log.select("gen", "max")
```

Запуск генетического алгоритма

Первый шаг генетического алгоритма - формирование начальной популяции. Он выполняется зарегистрированным в наборе инструментов методом (вызов) и его значение присваивается новой переменной.

Пример:

```
pop = toolbox.population(n=300)
```

Далее необходимо определить стратегию или модель функционирования генетического алгоритма. В модуле algorithms есть готовые реализации стратегий, которые можно использовать (Таблица 9), кроме этого можно запрограммировать работу генетического алгоритма по требуемой модели, используя операторы python (см. 2 пример). Для использования готовых стратегий необходимо учитывать следующие параметры:

- population - популяция,
- toolbox – набор инструментов,
- cxpb – вероятность кроссовера,

– mutpb – вероятность мутации,
– ngen – количество поколений,
– stats – статистика по ГА (необязательный параметр),
– halloffame – «зал славы», лучшие особи в поколениях(необязательный параметр),
– verbose – включать ли лог в статистику(необязательный параметр),
– mu – количество особей, выбираемых для следующего поколения,
– lambda_- количество потомков, порождаемых в каждом поколении.

Таблица 9 – Варианты моделей (стратегий) ГА

Функция	**Параметры**	**Описание**
deap.algorithms.eaSimple	(*population*, *toolbox*, *cxpb*, *mutpb*, *ngen*[, *stats*, *halloffame*, *verbose*])	Оба потомка заменяют родителей. Мутировать могут и родители и потомки.
deap.algorithms.eaMuPlusLambda	(population, toolbox, mu, lambda_, cxpb, mutpb, ngen[, stats, halloffame, verbose])	Только первый ребёнок добавляете в популяцию, второй отбрасывается. Потомок не подвергается мутации (мутируют родители). В (μ+λ)-стратегиях селекция производится из (μ+λ) особей - объединённой популяции родителей и потомков. Необходимо регистрировать функции: toolbox.mate(), toolbox.mutate(), toolbox.select() и toolbox.evaluate().
deap.algorithms.eaMuCommaLambda	(population, toolbox, mu, lambda_, cxpb, mutpb, ngen[, stats, halloffame, verbose])	Только первый ребёнок добавляете в популяцию, второй отбрасывается. Потомок не подвергается мутации (мутируют родители). В (μ, λ)-стратегиях в каждой итерации происходит генерация λ потомков, из которых выбирается μ особей. Необходимо регистрировать функции: toolbox.mate(), toolbox.mutate(), toolbox.select() и toolbox.evaluate().
deap.algorithms.eaGenerateUpdate	(*toolbox*, *ngen*[, *stats*, *halloffame*, *verbose*])	Алгоритм генерирует особи с помощью функции toolbox.generate() и изменяет / обновляет их с помощью toolbox.update(). Необходимо регистрировать функции: toolbox.generate(), toolbox.evaluate()

Пример:

4) Простая эволюционная стратегия, используются оба потомка.

```
pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40,
stats=stats, halloffame=hof, verbose=True)
```

5) 2) Стратегия (μ+λ).

```
MU, LAMBDA = 100, 100
pop, log = algorithms.eaMuPlusLambda(pop,toolbox,mu=MU,lambda_=LAMBDA,
cxpb=0.7, mutpb=0.3, ngen=NGEN, stats=stats, verbose=True, halloffame=hof)
```

Просмотр результатов функционирования генетического алгоритма

Результаты логов, статистики, зала славы можновывести на печать или на график (см. примеры ниже).

Пример реализации генетического алгоритма

1) Реализация ГА на базе модели eaSimple.

```
import random
```

```
import numpy
from deap import algorithms, base, creator, tools
def evalOneMax(individual):
    return sum(individual),

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_bool, 100)
toolbox.register("evaluate", evalOneMax)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
hof = tools.HallOfFame(1)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", numpy.mean)
stats.register("std", numpy.std)
stats.register("min", numpy.min)
stats.register("max", numpy.max)
pop = toolbox.population(n=300)
pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40,
stats=stats, halloffame=hof, verbose=True)
print(pop, log, hof)
```

2) Реализация ГА без модели (с помощью цикла), использование статистики и зала славы, вывод на график показателей функционирования ГА.

```
# pip install deap
# https://deap.readthedocs.io/en/master/
# Библиотека построения графиков
import matplotlib.pyplot as plt
# библиотека работы со случайными величинами
import random
# библиотека для работы с массивами
import numpy
# библиотека генетического алгоритма
from deap import base, creator,tools
#создаем классы FitnessMax на основе класса base.Fitness и Individual на
основе списка
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
#Инициализировать панель инструментов
toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,
toolbox.attr_bool, 10)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
# функция оценки
def evalOneMax(individual):
    return sum(individual),
#Зарегистрировать оператора оценки (фитнес-функцию)
toolbox.register("evaluate", evalOneMax)
# кроссовер
toolbox.register("mate", tools.cxTwoPoint)
# мутация
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
# отбор
toolbox.register("select", tools.selTournament, tournsize=3)
n=20
l=numpy.zeros(n, dtype=float)
ll=numpy.zeros(n, dtype=float)
```

```
# создаем объект история
history=tools.History()
# Decorate the variation operators
toolbox.decorate("mate", history.decorator)
toolbox.decorate("mutate", history.decorator)
def main():
    #генерируем популяцию, в скобках количество особей в популяции
    pop = toolbox.population(n=30)
    history.update(pop)
    # расчитываем массив приспособленности
    fitnesses = list(map(toolbox.evaluate, pop))
    # присваиваем значение приспособленности соответсвующим особям
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit
    fits = [ind.fitness.values[0] for ind in pop]
    #параметы скрещивания и мутации
    CXPB, MUTPB = 0.5, 0.2
    g = 0
    # Функционирование ГА
    while g < n-1:
        g = g + 1
        print("-- Поколение %i --" % g)
        # выбираем особи следующего поколения
        offspring = toolbox.select(pop, len(pop))
        # клонируем, чтобы образовывать пары
        offspring = list(map(toolbox.clone, offspring))
        # применяем кроссовер
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < CXPB:
                # скрещиваем 2 особи
                toolbox.mate(child1, child2)
                # удаляем скрещенные особи
                del child1.fitness.values
                del child2.fitness.values
        # применяем мутацию
        for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                # удаляем невутированный вариан особи
                del mutant.fitness.values
        # Проверка валидности значения приспособленности
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fits = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fits):
            ind.fitness.values = fit
        # заменяем популяцию на новое поколение
        pop[:] = offspring
        history.update(pop)
        # расчет приспособленности новой популяции
        fits = [ind.fitness.values[0] for ind in pop]
        length = len(pop)
        mean = sum(fits) / length
        sum2 = sum(x*x for x in fits)
        std = abs(sum2 / length - mean**2)**0.5
        l[g] = mean
        ll[g] = g
        print("  Минимальная приспособленность %s" % min(fits))
        print("  Максимальная приспособленность %s" % max(fits))
        print("  Среднее знаяение %s" % mean)
        print("  Std %s" % std)
# просмотр лучшей особи
        best_ind = tools.selBest(pop, 1)[0]
        print("Лучшая особь %s, %s" % (best_ind, best_ind.fitness.values))
```

```
main()
plt.plot(ll,l,linewidth=2.0)
```