ГУАП

КАФЕДРА №43

РАБОТА
ЗАЩИЩЕНА С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

профессор		Ю.А. Скобцов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

КУРСОВАЯ РАБОТА

ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ ОЦЕНКИ СТОИМОСТИ ПРОЕКТОВ В ПРОГРАММНОЙ ИНЖЕНЕРИИ

по дисциплине: ЭВОЛЮЦИОННЫЕ МЕТОДЫ ПРОЕКТИРОВАНИЯ ПРОГРАММНО-ИНФОРМАЦИОННЫХ СИСТЕМ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. 4536 _____ Д.Р.. Жиляев

подпись, дата — инициалы, фамилия

Санкт-Петербург 2019

РЕФЕРАТ

Отчет на 16 стр., 1ч, 10 источников, 2 рисунка

Перечень ключевых слов: COCOMO, роевый алгоритм.

Объекты исследования: реализация метода оценки стоимости программного проекта COCOMO с помощью роевого алгоритма на выборке данных.

Результат разработки: программа.

Содержание

1. Индивидуальное задание по варианту.
   - Разобраться в теоретическом описании математического метода оценки стоимости программного проекта –модели COCOMO.
   - Из приведенной таблицы экспериментальных данных (программных проектов НАСА) отобрать из 18 проектов в качестве обучающего множества 13 проектов.
   - В соответствии с вариантом курсового проекта, заданного таблицей определить тип используемого эволюционного алгоритма (генетический или роевой алгоритм, генетическое программирование), кодирование потенциального решения, вид ошибки в целевой функции, вид генетических операторов кроссовера, мутации и репродукции

№ варианта	Тип эволюционного алгоритма	Кодирование решения	Фитнес-функция
9	PA	Вещественный вектор	MD

2. Краткие теоретические сведения.

COCOMO

Одной из самых популярных моделей, используемых для оценки сложности проектируемого ПО, является COnstructive COst Model (COCOMO). Алгоритмическую модель стоимости можно построить с помощью анализа затрат и параметров уже разработанных проектов.

Алгоритмическим моделям присущи общие проблемы.

1. На ранней стадии выполнения проекта бывает сложно определить показатель размер при наличии информации только о системных требованиях. Несмотря на то, что оценка, основанная на функциональных, или объектных, точках, проще оценки размера кода, результаты тоже не всегда будут точными.

2. Оценка факторов, которые влияют на показатели, носит субъективный характер.

Значения этих показателей могут отличаться, если этим занимаются люди с разным опытом и квалификацией.

Основой для многих алгоритмических моделей оценки себестоимости является количество строк программного кода в созданной системе.

Оценку размера кода можно получить по аналогии с другими проектами путем преобразования функциональных точек n размер кода, сравнения размеров компонентов системы и сравнения с эталонными компонентами, а также просто па основе инженерной интуиции.

На размер окончательной системы могут повлиять решения, которые были приняты в процессе реализации проекта уже после утверждения начальной сметы.

Следует также помнить о большой вероятности значительных ошибок при раннем прогнозировании

себестоимости.

Наиболее точные оценки можно получить в том случае, если создаваемый продукт хорошо структурирован, модель учитывает интересы организации-заказчика, заранее определены язык программирования и необходимые аппаратные средства.

Точность результатов прогнозирования себестоимости также зависит от количества информации о создаваемой системе.

По ходу реализации проекта увеличивается количество информации, вследствие чего оценка себестоимости становится все более точной.

Для прогнозирования затрат применяется математическая формула, в которой учтены данные о размере проекта, количестве программистов, а также другие факторы и процессы.

Ядром модели является следующая формула $Ef=aLb$, где L - длина кода ПО в килостроках; Ef – оценка сложности проекта в человеко-месяцах; a и b - коэффициенты (параметры) модели, которые для различных типов ПО имеют различные значения.

Роевый алгоритм

В последнее десятилетие при решении задач оптимизации все шире используются новые методы, которые фактически примыкают к эволюционным вычислениям по своей идеологии и основаны на моделировании социального поведения живых организмов. К ним относятся, прежде всего, роевые алгоритмы (PSO–particle swarm optimization), которые, в основном, используются в численной оптимизации, и муравьиные алгоритмы (ACO – ant colony optimization), применяемые, как правило, при решении задач комбинаторной оптимизации (прежде всего на графах).

Роевые алгоритмы (РА), также как и эволюционные, используют популяцию особей – потенциальных решений проблемы и метод стохастической оптимизации, который навеян (моделирует) социальным поведением птиц или рыб в стае или насекомых в рое.

Аналогично эволюционным алгоритмам здесь начальная популяция потенциальных решений также генерируется случайным образом и далее ищется (суб)оптимальное решение проблемы в процессе выполнения РА. Первоначально в РА предпринята попытка моделировать поведение стаи птиц, которая обладает способностью порой внезапно и синхронно перегруппироваться и изменять направление полета при выполнении некоторой задачи.

Представленный основной роевой алгоритм часто называют глобальным РА (Global Best PSO), поскольку здесь при коррекции скорости частицы используется информация о положении достигнутого глобального оптимума, которая определяется на основании информации, передаваемой всеми частицами роя. В противоположность этому подходу часто используется локальный РА, где при коррекции скорости частицы используется информация, передаваемая только в каком-то смысле ближайшими соседними частицами роя

В данной работе будут использован глобальный роевый алгоритм, ниже представленный мета-код данного алгоритма:

```
Создание инициализация nx-мерного роя;
repeat
    for каждой частицы i=1,…,ns do
        // определить персональную лучшую позицию
        If f(xi) <f(yi) then
            yi= xi;
        end
         // определить глобальную лучшую позицию
        if f(yi) <f(y”i) then
            y*i= yi;
        end
    end
    for каждой частицы i=1,…,ns do
        //коррекция скорости
        vij(t+1)=vij(t)+c1*r1*[yij(t)-xij(t)]+ c2*r2*[y”ij(t)-xij(t)]
        //коррекция позиции
        xi(t+1)=xi(t)+vi(t+1)
    end
until критерий останова не выполнен;
```

Где vij(t) - j-ая компонента скорости i-ой частицы в момент времени t xij(t) - j-я координата позиции i-й частицы, c1 и c2 - положительные коэффициенты, регулирующие вклад когнитивной и социальной компонент, r1 и r2 - случайные числа из диапазона [0,1], которые генерируются в соответствии с нормальным распределением и вносят элемент случайности в процесс поиска. Кроме этого yij(t) - персональная лучшая позиция по j-й координате i-ой частицы, а y”ij(t) - лучшая глобальная позиция роя, где целевая функция имеет экстремальное значение.

Рассмотрим влияние различных составляющих при вычислении скорости частицы. Первое слагаемое в vij(t) сохраняет предыдущее направление скорости i-й частицы и может рассматриваться как момент, который препятствует резкому изменению направления скорости и выступает в роли инерционной компоненты. Когнитивная компонента c1*r1*[yij(t)-xij(t)] определяет характеристики частицы относительно ее

предыстории, которая хранит лучшую позицию данной частицы. Эффект этого слагаемого в том, что оно пытается вернуть частицу назад в лучшую достигнутую позицию. Третье слагаемое c2*r2*[y”ij(t)-xij(t)] определяет социальную компоненту, которая характеризует частицу относительно своих соседей. Эффект социальной компоненты в том, что она пытается направить каждую частицу в сторону достигнутого роем (или его некоторым ближайшим окружением) глобального оптимума.

Графически это наглядно иллюстрируется для двумерного случая, как это показано на нижеприведенном рисунке

(a) скорость во время t (b) скорость во время t+1

В качестве фитнесс-функции в данном случае нужно взять различие между реальными значениями стоимостей проектов и модельными значениями (оценками) стоимостей этих же проектов, которые вычислены согласно приведенной формуле с найденными с помощью ГА коэффициентами a и b. [,] a La Ub Это различие (расстояние между оценками) можно оценить по-разному - в различной метрике.

В качестве фитнесс-функции, взята метрика значений (Манхэттен – метрика городских кварталов), где это различие определяется с помощью следующей формулы

$$MD = \left(\sum_{i=1}^{n} |\text{Effort}_i - \text{Estimated Effort}_i| \right)$$

, Здесь Efforti - реальная (измеренная) стоимость i-го проекта в человеко месяцах и Estimated Efforti -модельная оценка того же проекта, вычисленная с помощью приведенной формулы с найденными путем применения ГА коэффициентами a и b.

Реализованный код на языке python:

```
# Манхэттен - метрика городских кварталов
def MD(x, T=Table):
    s = 0.0
    for el in T:
        s += abs(x[0] * el[0] ** x[1] - el[1])
    return s
```

3. Программа и результаты выполнения индивидуального задания с комментариями и выводами.

Основной код программы

```
import random
import numpy as np
import copy
from matplotlib import pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# ## Таблица

# In[22]:


Table = [
(902000, 30000, 1158000, 1340202),
(462000, 200000, 960000, 841616),
(465000, 190000, 790000, 850112),
(545000, 200000, 9098000, 949828),
(311000, 350000, 396000, 566580),
(675000, 29000, 984000, 1072609),
(128000, 26000, 189000, 326461),
(105000, 340000, 103000, 250755),
(215000, 310000, 285000, 443086),
(31000, 62000, 70000, 144563),
```

```
(42000, 190000, 90000, 199759),
(78000, 310000, 73000, 215763),
(21000, 280000, 50000, 112703),

(50000, 290000, 84000, 170887),
(786000, 350000, 987000, 1180378),
(97000, 270000, 156000, 268312),
(125000, 270000, 239000, 316864),
(1008000, 340000, 1383000, 1444587),
#     (1, 1.5),
#     (2, 12),
#     (3, 40.5),
#     (4, 96),
#     (5, 187.5),
#     (6, 324),
#     (7, 514.5),
#     (8, 768),
#     (9, 1093.5),
#     (10, 1500),
#     (11, 1996.5),
#     (12, 2592),
#     (13, 3295.5),
#     (14, 4116),
#     (15, 5062.5),
#     (16, 6144),
#     (17, 7369.5),
#     (18, 8748),
#     (19, 10288.5),
#     (20, 12000),
#     (21, 13891.5),
#     (22, 15972),
#     (23, 18250.5),
#     (24, 20736),
#     (25, 23437.5),
#     (26, 26364),
#     (27, 29524.5),
#     (28, 32928),
#     (29, 36583.5),
#     (30, 40500),

#     (25.9, 117.6),
#     (24.6, 117.6),
#     (7.7, 31.2),
#     (8.2, 36),
```

```
#     (9.7, 25.2),
#     (2.2, 8.4),
#     (3.5, 10.8),
#     (66.6, 352.8),
#     (7.5, 72),
#     (20, 72),
#     (6, 24),
#     (100, 360),
#     (11.3, 36),
#     (100, 215),
#     (20, 48),
#     (100, 360),
#     (150, 324),
#     (31.5, 60),
#     (15, 48),
#     (32.5, 60),
#     (19.7, 60),
#     (66.6, 300),
#     (29.5, 120),
#     (15, 90),
#     (38, 210),
#     (10, 48),
#     (15.4, 70),
#     (48.5, 239),
#     (16.3, 82),
#     (12.8, 62),
#     (32.6, 170),
#     (35.5, 192),
#     (5.5, 18),
#     (10.4, 50),
#     (14, 60),
#     (6.5, 42),
#     (13, 60),
#     (90, 444),
#     (8, 42),
#     (16, 114),
#     (177.9, 1248),
#     (302, 2400),
#     (282.1, 1368),
#     (284.7, 973),
#     (79, 400),
#     (423, 2400),
#     (190, 420),
#     (47.5, 252),
```

```
#     (21, 107),
#     (78, 571.4),
#     (11.4, 98.8),
#     (19.3, 155),
#     (101, 750),
#     (219, 2120),
#     (50, 370),
#     (227, 1181),
#     (70, 278),
#     (0.9, 8.4),
#     (980, 4560),
#     (350, 720),
#     (70, 458),
#     (271, 2460),
#     (90, 162),
#     (40, 150),
#     (137, 636),
#     (150, 882),
#     (339, 444),
#     (240, 192),
#     (144, 576),
#     (151, 432),
#     (34, 72),
#     (98, 300),
#     (85, 300),
#     (20, 240),
#     (111, 600),
#     (162, 756),
#     (352, 1200),
#     (165, 97),
#     (60, 409),
#     (100, 703),
#     (32, 1350),
#     (53, 480),
#     (41, 599),
#     (24, 430),
#     (165, 4178.2),
#     (65, 1772.5),
#     (70, 1645.9),
#     (50, 1924.5),
#     (7.25, 648),
#     (233, 8211),
#     (16.3, 480),
#     (6.2, 12),
```

```
#     (3, 38),
#     (902000, 115800),
#     (462000, 96000),
#     (465000, 790000),
#     (545000, 908000),
#     (311000, 396000),
#     (675000, 984000),
#     (128000, 189000),
#     (105000, 103000),
#     (215000, 285000),
#     (31000, 70000),
#     (42000, 90000),
#     (78000, 73000),

]
print(Table)
# Удалить повторяющиеся элементы
Table = list(set(Table))
# Граница 70/30
border = int(len(Table)*0.7)
test = Table[border:]
train = Table[:border]
print(test)
print(train)


# ## Фитнес функция

# In[27]:


# Манхэттен – метрика городских кварталов
# def MD(x, T=train):
#     s = 0.0
#     for el in T:
#         s += abs(x[0] * el[0] ** x[1] - el[1])
#         # print(x[0], x[1],el[0] ** x[1] - el[1], s)
#     return s
def MD(x, T=train,verbose=False):
    s = 0.0
    if verbose:
        print("REAL", "COCOMO", "RA", "MD")
        for el in T:
            F=x[0] * (el[0] ** x[1])*el[1]
```

```
            md=abs(F - el[2])
            s += md
            print(el[2], el[3], F, md)

    else:
        for el in T:
            s += abs(x[0] * (el[0] ** x[1])*el[1] - el[2])
    # print(x[0], x[1],el[0] ** x[1] - el[1], s)
    return s


# ## Генератор рандомных значений

# In[24]:


def generate(N, lb=0.0, rb=10.0):
    r = []
    for i in range(N):
         r.append((random.uniform(lb, rb), random.uniform(lb,
rb)))

    return r


# ## Роевой алгоритм

# In[25]:


# N - кол-во особей
# T - кол-во поколений
# c1,c2 - ветер
def ra(N=70, T=200, c1=1, c2=1):
    Y = np.array(generate(N))
    X = Y.copy()
    # X = Y
    ymax = copy.copy(Y[0])
    v = [0] * N

    x0 = []
    x1 = []

    for i in range(T):
```

```
            x0.append(i)
            x1.append(MD(ymax))
            for j in range(N):
                if MD(X[j]) < MD(Y[j]):
                    Y[j] = X[j]

                if MD(Y[j]) < MD(ymax):
                    ymax = Y[j]
                    # ymax = copy.copy(Y[j])
                    # print(MD(ymax))

            for j in range(N):
                   v[j] = v[j] + c1 * random.uniform(0.0, 1.0) *
(Y[j] - X[j]) + c2 * random.uniform(0.0, 1.0) * (ymax - X[j])
                X[j] = X[j] + v[j]

        x0.append(T+1)
        x1.append(MD(ymax))
        return [ymax,x0,x1]


    # In[28]:


    maximum, x0, x1 = ra(10, 10, c1=0.91, c2=0.81)
    print(maximum, MD(maximum, test, verbose=True))
    plt.plot(x0,x1)
```

Результаты выполнения

Входная таблица данных автоматически разбивается на обучающую и тестовую выборку в соотношении 70% на 30%. И удаляются повторяющиеся элементы.

Первый запуск программы:

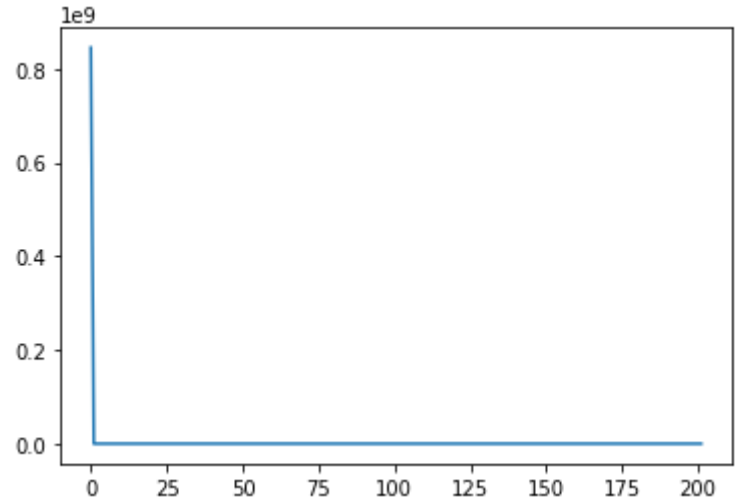
При запуске тестового скрипта выводится среднее значение относительной погрешности (MD) на тестовых данных при параметрах, найденных роевым алгоритмом с количеством особей 70 и количеством итераций 200, ниже будут приведены запуски программы на разных данных (из методического пособия, набор данных из проектов НАСА за 1971-1987 года и набор данных заранее известной формулы E=1.5*L^3, также набор данных из статьи,

которые задаются по формуле E=a*L^b*M, где M – влияние факторов, кроме длины кода программ)

```
T = [
    (21000, 50000),
    (50000, 84000),
    (786000, 987000),
    (97000, 156000),
    (125000, 239000),
    (1008000, 1383000),
    (902000, 115800),
    (462000, 96000),
    (465000, 790000),
    (545000, 908000),
    (311000, 396000),
    (675000, 984000),
    (128000, 189000),
    (105000, 103000),
    (215000, 285000),
    (31000, 70000),
    (42000, 90000),
    (78000, 73000),
]
```

Результат:

```
max [1.53363667 2.9929506 ]
Test 1.9773293976921595e+18
```



Второй запуск программы:

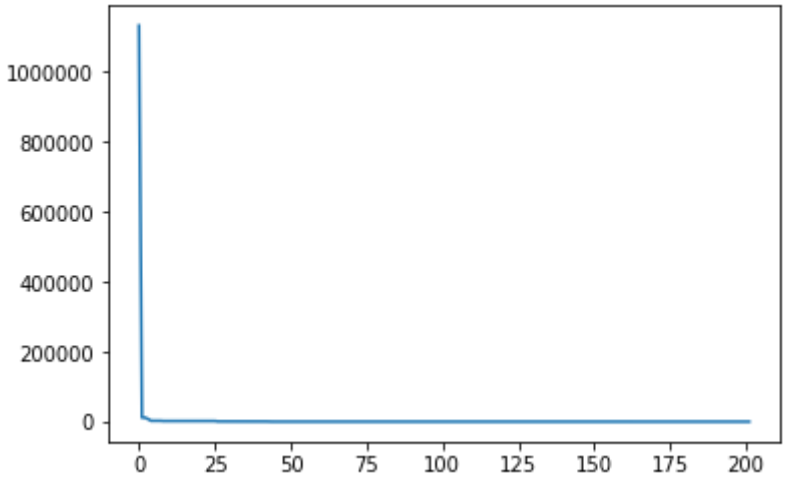
```
T = [
```

(25.9, 117.6),
(24.6, 117.6),
(7.7, 31.2),
(8.2, 36),
(9.7, 25.2),
(2.2, 8.4),
(3.5, 10.8),
(66.6, 352.8),
(7.5, 72),
(20, 72),
(6, 24),
(100, 360),
(11.3, 36),
(100, 215),
(20, 48),
(100, 360),
(150, 324),
(31.5, 60),
(15, 48),
(32.5, 60),
(19.7, 60),
(66.6, 300),
(29.5, 120),
(15, 90),
(38, 210),
(10, 48),
(15.4, 70),
(48.5, 239),
(16.3, 82),
(12.8, 62),
(32.6, 170),
(35.5, 192),
(5.5, 18),
(10.4, 50),
(14, 60),
(6.5, 42),
(13, 60),
(90, 444),
(8, 42),
(16, 114),
(177.9, 1248),
(302, 2400),
(282.1, 1368),
(284.7, 973),

(79, 400),
(423, 2400),
(190, 420),
(47.5, 252),
(21, 107),
(78, 571.4),
(11.4, 98.8),
(19.3, 155),
(101, 750),
(219, 2120),
(50, 370),
(227, 1181),
(70, 278),
(0.9, 8.4),
(980, 4560),
(350, 720),
(70, 458),
(271, 2460),
(90, 162),
(40, 150),
(137, 636),
(150, 882),
(339, 444),
(240, 192),
(144, 576),
(151, 432),
(34, 72),
(98, 300),
(85, 300),
(20, 240),
(111, 600),
(162, 756),
(352, 1200),
(165, 97),
(60, 409),
(100, 703),
(32, 1350),
(53, 480),
(41, 599),
(24, 430),
(165, 4178.2),
(65, 1772.5),
(70, 1645.9),
(50, 1924.5),

```
    (7.25, 648),
    (233, 8211),
    (16.3, 480),
    (6.2, 12),
    (3, 38),

]
Результат:
Макс [1.62922064 2.97350397]
Тест 1492510556.1298385
```



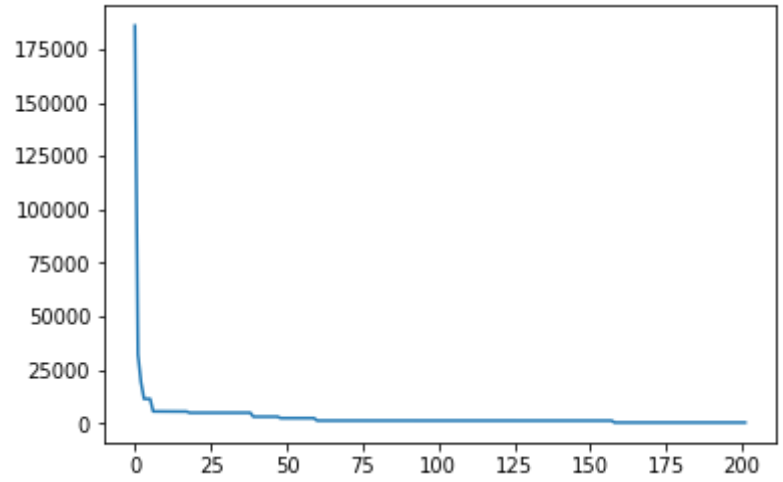
Третий запуск программы:
N = 70
T = 200
C1 = 1
C2 = 1
Table =[
(1, 1.5),
(2, 12),
(3, 40.5),
(4, 96),
(5, 187.5),
(6, 324),
(7, 514.5),
(8, 768),
(9, 1093.5),
(10, 1500),
(11, 1996.5),
(12, 2592),
(13, 3295.5),
(14, 4116),

(15, 5062.5),
(16, 6144),
(17, 7369.5),
(18, 8748),
(19, 10288.5),
(20, 12000),
(21, 13891.5),
(22, 15972),
(23, 18250.5),
(24, 20736),
(25, 23437.5),
(26, 26364),
(27, 29524.5),
(28, 32928),
(29, 36583.5),
(30, 40500),
]

Вывод

max: [1.45719077 3.00921002]

Test: 235.1137423227117



Значения хороши, так как наши идеальные параметры, которые мы загадали равны соответственно 1.5 и 3.0.

Четвертый запуск программы:

```
Table = [
(902000, 30000, 1158000, 1340202),
(462000, 200000, 960000, 841616),
(465000, 190000, 790000, 850112),
(545000, 200000, 9098000, 949828),
(311000, 350000, 396000, 566580),
(675000, 29000, 984000, 1072609),
(128000, 26000, 189000, 326461),
```

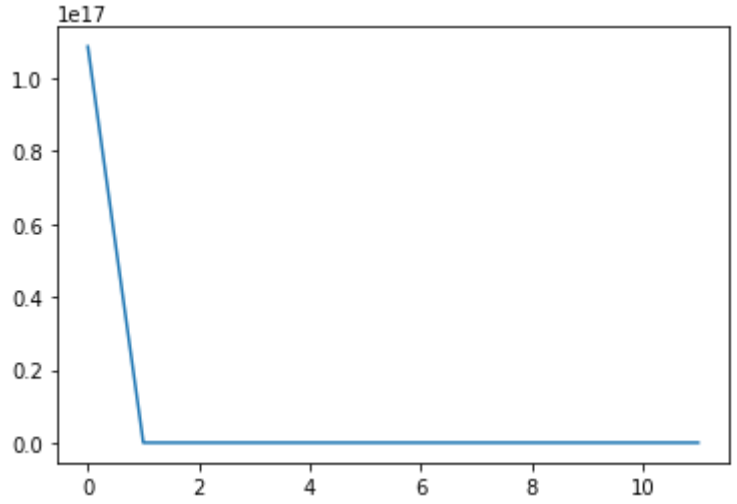
```
(105000, 340000, 103000, 250755),
(215000, 310000, 285000, 443086),
(31000, 62000, 70000, 144563),
(42000, 190000, 90000, 199759),
(78000, 310000, 73000, 215763),
(21000, 280000, 50000, 112703),

(50000, 290000, 84000, 170887),
(786000, 350000, 987000, 1180378),
(97000, 270000, 156000, 268312),
(125000, 270000, 239000, 316864),
(1008000, 340000, 1383000, 1444587),
]
```

Результат программы на обучении и тестирование вышеприведенных данных (в таблице выведены столбцы: килостроки в тестовой выборке, реальная сложность в тестовой выборке, сложность по COCOMO в тестовой выборке, сложность полученная при помощи PA, ошибка MD). Параметры Роевого алгоритма: N=10, T=10, c1=0.91, c2=0.81

REAL	COCOMO	RA	MD
96000	841616	184504.69	775495.30
84000	170887	235130.89912258205	151130.89912258205
285000	443086	273559.4914205924	11440.508579407586
396000	566580	315548.5792327072	80451.42076729279
103000	250755	287804.95945172745	184804.95945172745
1158000	1340202	28771.896219006587	1129228.1037809935

Также приведен график среднего значения относительной погрешности MD:



Вывод

В данной работе был реализован метод оценки стоимости программного проекта COCOMO с помощью роевого алгоритма.

Список литературы

1. Соммервилл, Иан. Инженерия программного обеспечения, 6-е издание.: Пер. с англ.- М. :Издательский дом "Вильямс", 2002.- 624 с.
2. Ю. А. Скобцов, Д. В. Сперанский. Эволюционные вычисления – М.: Национальный Открытый Университет “ИНТУИТ”, 2015. - 331с.
3. Ю. А. Скобцов. Электронный конспект по дисциплине « Генетические алгоритмы в разработке программно-информационных систем».- ГУАП.-2017.
4. Ю. А. Скобцов. Основы эволюционных вычислений – Учебное пособие.- Донецк: ДонНТУ, 2008. - 326 с.
5. Ю. А. Скобцов, Е. Е. Федоров. Метаэвристики – Монография. Донецк: Ноулидж, 2013. - 426 с.
6. А.Л. Рыжко, Н.М. Лобанова, Н.А. Рыжко, Е.О. Кучинская. Экономика информационных систем. М.: Финуниверситет, 2014. - 204 с.
7. Barry Boehm, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice—Hall, 2000
8. Курбатова Е.А. MATLAB 7. Самоучитель. Издательство: Вильямс. Год издания: 2005г. - 256 стр.
9. В. Потемкин. Вычисления в среде MATLAB. Диалог-МИФИ. 2004.
10. Алексеев Е.Р., Чеснокова О.В. MATLAB 7. Самоучитель. Издательство "НТ Пресс" 2006г. - 464 стр.