Федеральное государственное автономное образовательное учреждение
высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**
Факультет экономических наук

Отчет по проделанной работе

Стохастические методы оптимизации и их приложения к задачам финансовой эконометрики

по направлению подготовки «Экономика»
образовательная программа «Экономика»

Выполнил:
Студент группы БЭК 201
Зайцев Александр Алексеевич

Научный руководитель:
Борзых Дмитрий Александрович

Москва — 2023

Содержание

1 Метод имитации отжига

1.1 Игра в шахматы

1.1.1 Постановка задачи

В данной задаче рассматривается следующая функция $H : S \to R$, где H - функция, которая минимизируется, S - множество на котором происходит оптимизация.

В задаче "игры в шахматы" по условию задается шахматная доска и 8 шахматных фигур - ферзей. Задача состоит в том, чтобы найти такое расположение ферзей на доске, при котором ни один не "атакует", то есть не достигает по диагонали, вертикали или горизонтали до другого ферзя.

Для решения подобной задачи введено понятие "количество ударов", под которым понимается количество ферзей, которые могут быть атакованными другими ферзями, учитывая "удар" сквозь друг другого ферзя. Стартовым расположением является вектор $[1, 2, 3, 4, 5, 6, 7, 8]$, что представлено на рисунке 1. Индекс заданного вектора является положением по вертикали, а значение - по горизонтали. В рамках стартового расположения наблюдается 56 ударов.

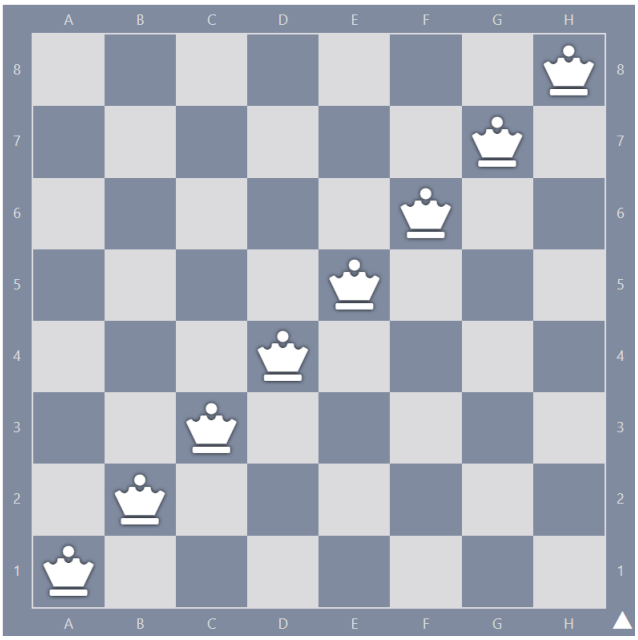

Рис. 1: Начальное расположение фигур

H

1.1.2 Реализации алгоритма

Для оптимизации данной задачи необходимо минимизировать количество ударов, что выступает целевой функцией для метода имитации отжига. В рамках алгоритма исполняется цикл до достижения минимального значения количества ударов - оптимум данной задачи находится в точке 0. Исполняется следующий алгоритм:

1. Определение функции для оптимизации $H(s), s \in S$ - для подсчета количества "ударов"используются циклы, относительно которых рассматриваются любые различные пары элементов: если для двух любых ферзей совпадает расположение по оси, представленной в виде значения вектора, или фигуры расположены на одной диагонали, тогда засчитывается 1 "удар"для каждого из двух ферзей (листинг 1: Функция "ударов"ферзей).

2. (листинг 2: Метод имитации отжига для функции "ударов"ферзей) Инициализация T_0 - начальная температура, $point \in S$ - начальное расположение, $alpha$ - коэффициент снижения температуры.

3. Случайное изменение расположения двух ферзей. Можно заметить, что в качестве решения ферзи не могут стоять на одной вертикали и горизонтали - это значит их индексация расположения не может быть одинаковой, следовательно, для нахождения отимума достаточно переставлять местами элементы текущего вектора (строка 8 - 13).

4. Рассчет количества ударов в текущем расположении $H(s_{t+1})$ (строка 14).

5. Если количество ударов уменьшилось, то принять новое заданное расположение и уменьшить температуру отжига (строка 15-17). В другом случае с заданной вероятностью $p = \exp \frac{H(s_t)-H(s_{t+1})}{T_t}$ принять новое положение (иначе остается предыдущее значение. По уменьшении температуры вероятность принятия нового положения снижается, строка 18-22).

6. Повторить шаги 1-4 до сходиомсти.

Листинг 1: Фунция "ударов"ферзей

```
function strike = queen(myRange)
strike = 0;
for i=1:length(myRange)
    for j=1:length(myRange)
        if i~=j
            if myRange(i)==myRange(j) || abs(myRange(i)-myRange(j)) ==
               abs(j-i)
                strike = strike + 1;
            end
        end
    end
end
end
```

Листинг 2: Метод имитации отжига для функции "ударов"ферзей

```
function [point, number_oper] = annealing_queen(point, T, alpha)
cur_min = 1;
number_oper = 0;
while cur_min > 0.1
    number_oper = number_oper + 1;
    new_point = point;
    cur_min = queen(point);
    f = randi([1, length(point)]);
    s = randi([1, length(point)]);
    f_n = new_point(f);
    s_n = new_point(s);
    new_point(f) = s_n;
    new_point(s) = f_n;
    cur_min_2 = queen(new_point);
    if cur_min_2-cur_min<0
        point = new_point;
        T = alpha * T;
    elseif cur_min_2 - cur_min >= 0
        pr = exp(-(cur_min_2 - cur_min)/T);
        if binornd(1, pr) == 1
            point = new_point;
            T = alpha * T;
        end
    end
end
```

```
end
```

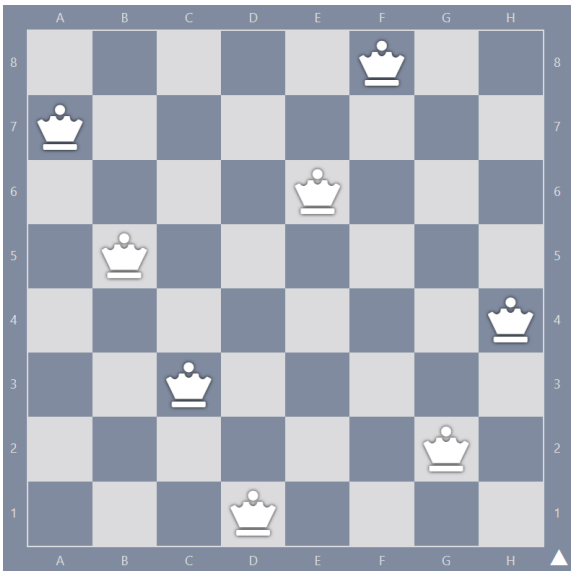

Рис.1.1 Пример №1

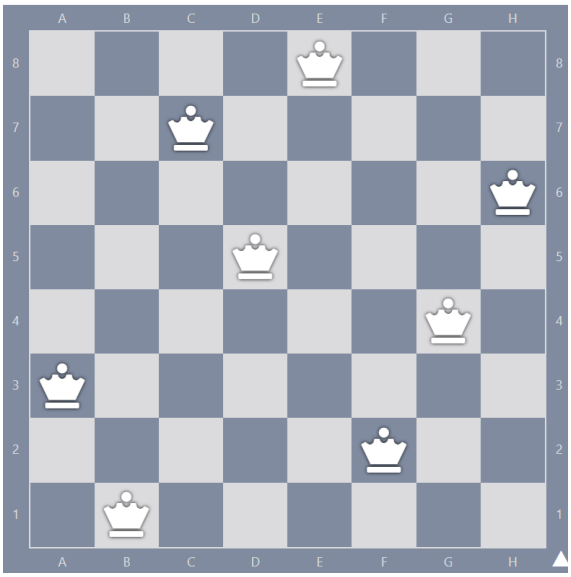

Рис.1.2 Пример №2

Листинг 3: Пример работы метода имитации отжига для ”игры в шахматы”

```
annealing_queen([1 2 3 4 5 6 7 8], 100, 0.95)
ans = [7 5 3 1 6 8 2 4]
```

На рис. 1.1 Пример №1, №2.

1.2 Оптимизация нелинейных функций

1.2.1 Постановка задачи

Методом имитации отжига можно также минимизировать непрерывные функции, к примеру, в рамках данного раздела рассматривается сложная функция для оптимизации градиентным спускок функция, так как имеет периодичные локальные минимумы:

$$f(x) = x^2 \cdot (2 + |sin(8x)|)$$

Листинг 4: Непрервная функция оптимизации

```
function z = example_funct(x)
    z = x^2 * (2 + abs( sin(8*x) ));
end
```

1.2.2 Реализации алгоритма

Оптимизация метода имитации отжига в рамках "Игры в шахматы" и нелинейных функций схожа. Один из способов решения данной задачи является прибавление к текущему расположению (координате) случайного сдвига. Полный алгоритм:

1. Инициализация T - начальная температура, $point$ - начальное расположение, $alpha$ - коэффициент снижения температуры.
2. Прибавление случайную величину для поиска нового минимума $point := point + \xi \ \ , \xi \sim U(0,1)$ (строка 5-7).
3. Расчет значения функции в новой точки $f(point)$ (строка 9).
4. Если значение функции уменьшилось, то принять новое заданное расположение и уменьшить температуру отжига (строка 10-12). В другом случае с заданной вероятностью принять новое положение или остаться в предыдущем. (по уменьшении температуры вероятность принятия нового положения снижается, строка 13-17).
5. Повторить шаги 1-4 до сходиомсти.

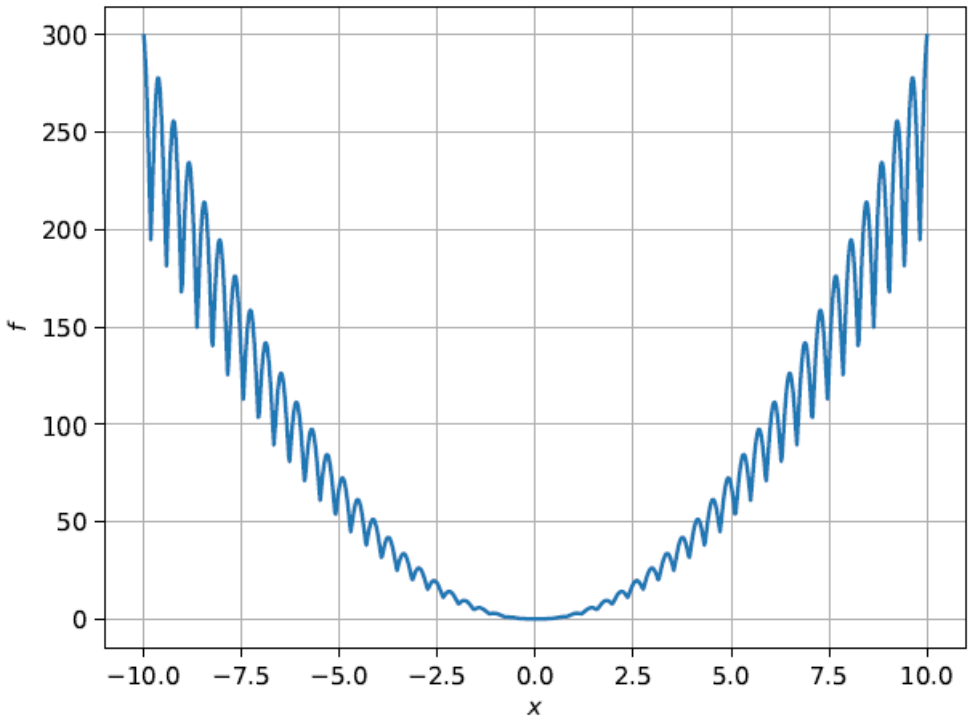

Рис. 2: График функции $f(x) = x^2 * (2 + |sin(8x)|)$

В рамках оптимизации при начальной точке $x = 100$ минимум функции достигается в среднем за 50-100 тысяч итераций.

Листинг 5: Метод имитации отжига для непрерывной функции

```
function [point, cur_min, number_oper] = annealing_funct(point, T, alpha)
cur_min = 1;
number_oper = 0;
while cur_min > 1.0000e-09
    xsi = unifrnd(-1, 1);
    number_oper = number_oper + 1;
    new_point = point + xsi;
    cur_min = example_funct(point);
    potential_min = example_funct(new_point);
    if potential_min-cur_min<0
        point = new_point;
        T = alpha * T;
    elseif potential_min-cur_min >= 0
        pr = exp(-(potential_min - cur_min)/T);
        if binornd(1, pr) == 1
            point = new_point;
            T = alpha * T;
        end
```

```
    end
end
end
```

Листинг 6: Пример работы метода имитации отжига для непрерывной функции

```
[~, cur_min, ~] = annealing_funct(50, 100, 0.95)
cur_min = 4.0030e-12
```

2 Метод роения частиц

2.0.1 Постановка задачи

Для нахождения оптимума нелинейных функций также можно использовать коллективные методы поиска решений, в частности, метод роения частиц. Рассматривается следующая задача: $\min\limits_{x \in R^n} F(x) = F(x^*)$.

В рамках нелинейных функций в данном разделе выступают функции Розенброка (1) (рис 3) и функции из учебника Шведова А.С (2):

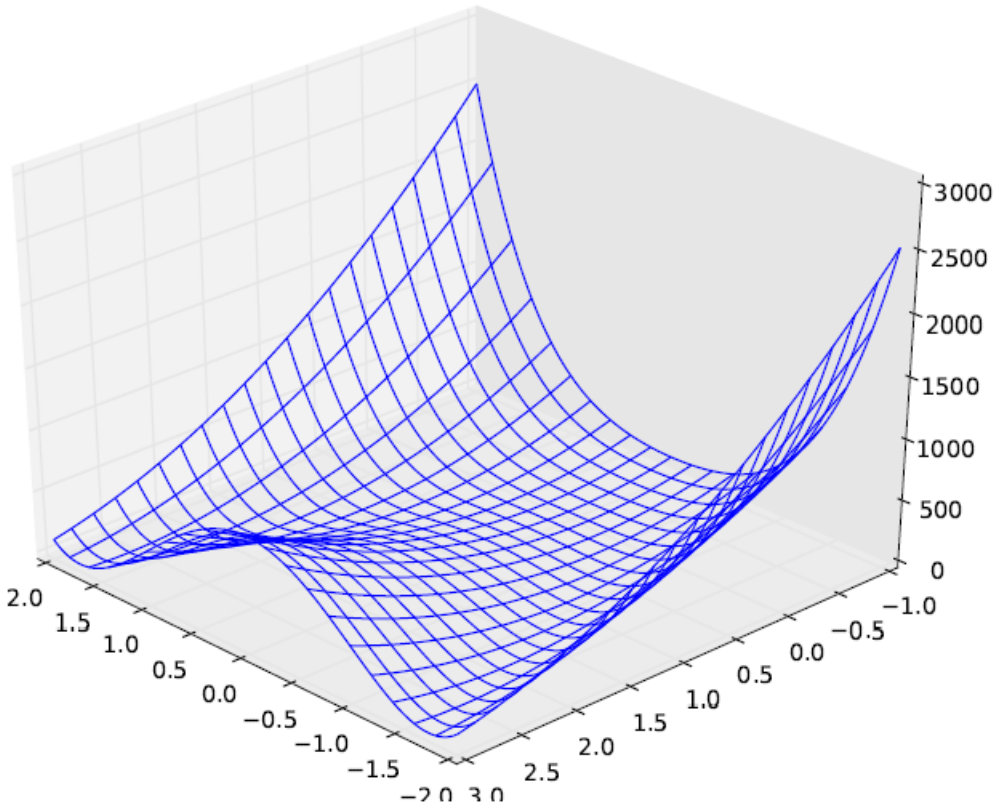

Рис. 3: График функции Розенброка

Функция Розенброка - минимум достигается в точке $(1, 1)$:

$$f(x, y) = (1 - x)^2 + 100 \cdot (y - x)^2 \tag{1}$$

Функция из учебника Шведова А.С. - минимум в точке $(0.5, 0.25, 0.25)$:

$$f(x, y, z) = 0.01 \cdot (x - 0.5)^2 + |x^2 - y| + |x^2 - z| \tag{2}$$

Листинг 7: Реализация функции Розенброка

```
    x = mat(:,1).';
    y = mat(:,2).';
    z = (ones(1,length(x))-x).^2+100*(y-x.^2).^2;
end
```

Листинг 8: Реализация функции из учебника Шведова А.С.

```
function f = Shvedov(mat)
    x = mat(:,1).';
```

```
3     y = mat(:,2).';
4     z = mat(:,3).';
5     f = 0.01 * (x - ones(1,length(x)) * 0.5).^2 + abs(x.^2 - y) + abs(x.^2
          - z);
6 end
```

2.0.2 Реализации алгоритма

Алгоритм метода роения частиц:

1. Генерация M равномерно распределенных от 0 до 1 частиц (точек), которые зависят от времени: $G = (x_1, x_2, .., x_M)$, $x_i(t) \in R^m, i \in (0, 1, ..., M), t \in (1, ..., T)$ (строка 3).

   Для каждой точки генерируется случайная скорсть в начальный момент времени $v_i(t) \in R^m, i \in (0, 1, ..., M), t \in (1, ..., T)$ (строка 4).

2. Вычисление значения точки в следующий момент времени $x_i(t+1) = x_i(t) + v_m(t)$, где

$$v_m(t) = \alpha v_m(t-1) + \beta\xi_2 \cdot (\tilde{x}_m(t) - x_m(t)) + \gamma\xi_3(\tilde{x}(t) - x_m(t))$$

   где $\alpha, \beta, \gamma \in (0, 1)$,

   $\xi_2, \xi_3 \sim i.i.d\ U(0, 1)$ (строка 27 - расчет скорости для t>0, строка 28 - расчет координат для t>0)

   $f(x(t)) = \tilde{x}(t)$ - глобальный минимум за всё время

   $f(x_i(t)) = \tilde{x}_i(t), \in (0, 1, ..., M)$ - локальный минимум которая достигала точка за все время.

3. Для каждой точки необходимо запомнить её локальный минимум, а также глобальный минимум из всех локальный минимумов. (строка 6-7 - стартовый локальный минимум, строка 34-35 расчет для t>0 локальных минимумов, строка 10-11 расчет стартового глобального минимума, строка 10-11 расчет для t>0 глобальных минимумов).

4. Повторить шаги 2-6 до сходимости.

Результаты: Метод роения частиц находит минимум для функции Розенберга менее чем за 10 итераций, а для функции из учебника Шведова А.С. достигается локальный минимум в точке x_{loc}, где $F(x_{loc}) = 0.0025$ менее чем за 10 итераций, но глобальный минимум иногда достигается более чем за 1000 тысяч итераций.

Листинг 9: Метод роения частиц

```
function [glob_min_funct, glob_min_cord] = ROI_alogorithm(funct,
   number_of_bees, alpha, beta, gamma, dim, L)

all_bees = unifrnd(-1, 1, number_of_bees , dim); %

V = unifrnd(-1, 1, number_of_bees , dim); %

local_min_coord = all_bees;
local_min_funct = funct(all_bees).';


glob_min_funct = min(local_min_funct);
glob_min_cord =
   local_min_coord(find(glob_min_funct==min(local_min_funct)),:);

num_op = 0;
while num_op < L
    num_op = num_op + 1;

    for ii = 1:length(all_bees)
        number_bee = ii;
        bee = all_bees(number_bee, :);
        V_bee = V(number_bee, :);
        local_coord_bee = local_min_coord(number_bee, :);

        xsi = unifrnd(0, 1);
        nu = unifrnd(0, 1);


        V_bee = V_bee.*alpha + beta .* xsi .* (local_coord_bee - bee) +
           gamma .* nu .* (glob_min_cord - bee);
        bee = bee + V_bee;

        V(number_bee, :) = V_bee;
```

```
        all_bees(number_bee, :) = bee;
        cur_place_funct = funct(bee).';
        if cur_place_funct < local_min_funct(number_bee, :)
            local_min_funct(number_bee, :) = cur_place_funct;
            local_min_coord(number_bee, :) = bee;
        end

        if cur_place_funct < glob_min_funct
            glob_min_funct = cur_place_funct;
            glob_min_cord = bee;
        end
    end
end
```

Листинг 10: Пример реализации метода роения частиц для функции Розенброка

```
 [glob_min_funct, glob_min_cord] = ROI_alogorithm(@Rosenbrog, 100, 0.95,
    0.2, 0.2, 5, 1000)
glob_min_funct = 1.8094e-22
glob_min_cord = 1.0000  1.0000  -1.1849   1.1692   1.5853
```

Листинг 11: Пример реализации метода роения частиц для функции из учебника А.С. Шведова

```
[glob_min_funct, glob_min_cord] = ROI_alogorithm(@Shvedov, 100, 0.95, 0.2,
   0.2, 5, 100000)
glob_min_funct = 0.0025
glob_min_cord = 1.2310  1.0133   1.1317  -0.2259  -0.5253
```

Пример для функции из учебника А.С. Шведова не сошелся - точка находится в локальном минимуме и для нахождения глобального минимума необходимо больше итераций в рамках данной функции.

3 Генетический алгоритм

3.1 Поиск минимума непрерывной функции

3.1.1 Постановка задачи

Для нахождения оптимума многомерных нелинейных функций можно воспользоваться коллективным методом оптимизации - генетическим алгоритмом. Рассматриваются функции вида $F : R^n \to R$, $F(x) \geq 0 \; \forall x \in R^n$, к примеру:

$$F(x) = \sum_{k=1}^{n} x_k^2(1 + |sin(100 \cdot x_k)|), x \in R^n$$

Листинг 12: Непрерывная неотрицательная функция для генетического алгоритма

```
1 function z = Genetic_function(x)
2 z = sum(x.*x.*(1 + abs(sin(100*x))));
3 end
```

Рассматривается $n = 5$. Данная функция имеет большое количество локальных минимумов, глобальный минимум достигается в точке $x = 0$.

3.1.2 Реализация метода

Функция активации - мера приспособленности. Чем больше мера приспособленности тем лучше особь.

В качестве функции активации в данной задаче рассматривается:

$$Fit(x) = \frac{1}{1 + F(x)}$$

Алгоритм:

1. Инициализуется M - количество особей, M_C - количество отбрасываемых особей в дальнейшем, dim - размерность функции.

2. Создание M особей $x \in R^n, n = 5$ (строка 2).

3. Для каждой особи рассчитывается её функция приспособленности и сортируется по убыванию (строка 7-10).

4. Особи с номером $3 \leq m \leq (M - M_C)$ заменяются на скрещенные особи. С вероятностью $p_i = \frac{Fix(x_i)}{\sum_{k=3}^{M-M_C} Fix(x_k)}$ новые особи будут иметь часть генов особи с номером i (строка 13-14 расчет вероятностей, строка 18-19 - скрещивание).

5. Особи с номерами $(M - M_C) \leq m \leq M$ заменяются на случайные (строка 23).

6. Повтор 3-4 до сходимости.

Функция находит глобальный минимум приблизительно за 50-100 итераций.

Листинг 13: Генетический алгоритм для непрерывной неотрицательный функции

```
1  function [funct_value, coords, num_op] = Genetic_algorithm(M, MC, L, dim)
2      gens = unifrnd(-1, 1, dim, M, L);
3      num_op = 0;
4      while num_op < L
5          new_gen = ones(dim, M);
6          num_op = num_op + 1;
7          adaptation = ones(1, M)./(ones(1, M)+Genetic_function(gens));
8          sort_adaptation = sort(adaptation, 'descend');
9          [~, refidx] = sort(adaptation, 'descend');
10         gens = gens(:,refidx);
11         coords = gens(:, 1);
12         funct_value = Genetic_function(coords);
13         sort_adaptation = sort_adaptation(:, 3:M-MC);
14         prob = sort_adaptation./sum(sort_adaptation);
15         gens_2_M_MC = gens(:, 3:M-MC);
16         new_gen_2_M_MC = gens_2_M_MC;
17
18         for i = 1:size(new_gen_2_M_MC, 1)
19             new_gen_2_M_MC(i, :) = randsample(gens_2_M_MC(i,:),
                   size(gens_2_M_MC, 2), true, prob);
20
21         end
22         new_gen(:, 2:M-MC-1) = new_gen_2_M_MC;
23         new_gen(:, M-MC:M) = unifrnd(-1, 1, dim, MC+1);
24         new_gen(:, 1:2) = gens(:, 1:2);
25         gens = new_gen;
```

```
26      end
27  end
```

Листинг 14: Пример работы генетического алгоритма для непрерывной неотрицательный функции

```
1  Genetic_algorithm(1000, 200, 1000, 5)
2  ans = 2.6373e-04
```

3.2 Поиск минимума задачи разделения множества

3.2.1 Постановка задачи

Дано конечное множество натуральных чисел A. Задача состоит в рабиении множества на 2 подмножества $A = K_1 \cup K_2$ так, чтобы пересечение этих подможеств образовало пустое множество, а также минимизировать

$$H(K_1, K_2) = \left| \sum_{a_k \in K_1} a_k - \sum_{a_m \in K_2} a_m \right|$$

Листинг 15: Функция для задачи разделения на множества

```
1  function z = Subsets(x)
2  A = 1:size(x,1);
3  B = repmat(A.', 1, size(x,2));
4  K1 = B.*x;
5  K2 = B.*(1 - x);
6  z = abs(sum(K_1, 1) - sum(K_2, 1));
7  end
```

3.2.2 Реализация метода

Алгоритм реализации тот же, что и в поиске минимума непрерывной функции генетическим алгоритом. Главное отличие состоит в генерации целых чисел (строка 2) и в функции оптимизации.

Листинг 16: Генетический алгоритм для задачи разделения множества

```
1  unction [funct_value, coords, num_op] = Genetic_algorithm_2(M, MC, L, dim)
2      gens = round(unifrnd(0, 1, dim, M));
```

```
    num_op = 0;
    while num_op < L
        new_gen = ones(dim, M);
        num_op = num_op + 1;
        adaptation = ones(1, M)./(ones(1, M)+Subsets(gens));
        sort_adaptation = sort(adaptation, 'descend');
        [~, refidx] = sort(adaptation, 'descend');
        gens = gens(:,refidx);
        coords = gens(:, 1);
        funct_value = Subsets(coords);
        sort_adaptation = sort_adaptation(:, 3:M-MC);
        prob = sort_adaptation./sum(sort_adaptation);
        gens_2_M_MC = gens(:, 3:M-MC);
        new_gen_2_M_MC = gens_2_M_MC;
        for i = 1:size(new_gen_2_M_MC, 1)
            new_gen_2_M_MC(i, :) = randsample(gens_2_M_MC(i,:),
                size(gens_2_M_MC, 2), true, prob);

        end
        new_gen(:, 2:M-MC-1) = new_gen_2_M_MC;
        new_gen(:, M-MC:M) = round(unifrnd(0, 1,dim, MC+1));
        new_gen(:, 1:2) = gens(:, 1:2);
        gens = new_gen;
    end
end
```

Листинг 17: Пример реализации генетического алгоритма для задачи разделения множества

```
[funct_value, ~, ~] = Genetic_algorithm_2(1000, 200, 100, 5000)
funct_value = 4

[funct_value, ~, ~] = Genetic_algorithm_2(1000, 200, 200, 5000)
funct_value = 0
```

3.3 Поиск минимума задачи линейного программирования

3.3.1 Постановка задачи

Дана технологичечская матрица , которая отражает сколько ресуров нужно потратить для производства одного товара (по строкам отмечены товары, по колонкам ресурсы). Также существуют вектор запасов ресурсов b и вектор цен товаров c. Необходимо найти наиболее оптимальный выбор производства товара с точки зрения максимизации прибыли с учетом того, что количиства некоторых товаров могут быть только целыми числами.

Листинг 18: Функция прибыли

```
1 function [z, x_new] = Genetic_function_3(x, a, b, c)
2 gens_sum = a.' * x;
3 leftover = b.' - gens_sum;
4 [row, columns] = find(leftover < 0);
5 columns = unique(columns.');
6 x(:,columns.') = zeros(size(x, 1), length(columns));
7 x_new = x;
8 z = c * x_new;
9 end
```

В рамках функции расчитывается количество потраченных ресурсов (строка 2), затем определяется сходится ли это с запосом b - если запосов меньше чем потраченных ресурсов, то данный вектор товаров зануляется (строка 3-6), затем расчитывается прибыль по каждому вектору (строка 8).

3.3.2 Реализация метода

Алгоритм реализации тот же, что и в поиске минимума непрерывной функции генетическим алгоритмом. Отличие заключается в том, что функция больше не является неотрицательной, следовательно, в качестве функции приспособленности была реализована функция прибыли (строка 8). Кроме того была изменена процедура генерации особей. Добавлена возможность ввода ограничения на целочисленность количества товаров.(int).

Листинг 19: Генетический алгоритм для задачи линейного программирования

```
1  function [funct_value, coords, num_op] = Genetic_algorithm_3(M, MC, L, a,
       b, c, int)
2      dim = size(a, 1);
3      gens = Genetic_generation(M, a, b, int);
4      num_op = 0;
5      while num_op < L
6          new_gen = ones(dim, M);
7          num_op = num_op + 1;
8          adaptation = Genetic_function_3(gens, a, b, c);
9          sort_adaptation = sort(adaptation, 'descend');
10         [~, refidx] = sort(adaptation, 'descend');
11         gens = gens(:,refidx);
12         coords = gens(:, 1);
13         funct_value = Genetic_function_3(coords, a, b, c);
14         sort_adaptation = sort_adaptation(:, 3:M-MC);
15         sort_adaptation = sort_adaptation + unifrnd(0, 10 * eps,
               size(sort_adaptation, 1));
16         prob = sort_adaptation./sum(sort_adaptation);
17         gens_2_M_MC = gens(:, 3:M-MC);
18         new_gen_2_M_MC = gens_2_M_MC;
19         prob(prob<0) = 0;
20         for i = 1:size(new_gen_2_M_MC, 1)
21             new_gen_2_M_MC(i, :) = randsample(gens_2_M_MC(i,:),
                   size(gens_2_M_MC, 2), true, prob);
22         end
23         new_gen(:, 2:M-MC-1) = new_gen_2_M_MC;
24         new_gen(:, M-MC:M) = Genetic_generation(MC+1, a, b, int);
25         new_gen(:, 1:2) = gens(:, 1:2);
26         gens = new_gen;
27     end
28 end
```

В рамках функции для генерации товаров расчитывается максимальное количество каждого товара, которое можно приобрести за данное количество ресурсов (строка 2-3). Далее генерируется векторы значений и на основе булевых масок выбирается целочисленное или вещественное количество товаров (строка 6-8).

Листинг 20: Функция генерации товаров для генетического алгоритма

```
function z = Genetic_generation(M, a, b, int)
    leftover_res = bsxfun(@rdivide, b, a);
    max_res = min(leftover_res.');
    z = ones(size(a, 1), M);
    for i = 1:size(a, 1)
        value_part = unifrnd(0, max_res(i), 1, M) * (1 - sum(ismember(int,
           i )));
        int_part = sum(ismember(int, i)) * randi([0 floor(max_res(i))],1,M);
        latent_z = value_part + int_part;
        z(i,:) = latent_z ;
    end
```

Листинг 21: Пример генетического алгоритма для задачи линейного программирования

```
M = 2000;
M_c = 1000 ;
L = 5000;

%1
a = eye(4).';
b = [1, 2, 3, 4];
c = [1, 1, 1, 1];
int = [1, 3];
[funct_value, coords, num_op] = Genetic_algorithm_3(M, M_c, L, a, b, c,
   int);
disp(b) %[1, 2, 3, 4]
disp(coords.') %[1.0000   1.9999    3.0000    3.9993]

%2
a = [57 3 29 47 71; 12 17 31 6 73; 5 43 37 59 79; 7 5 41 61 9; 11 23 43 67
   11];
b = [29 77 91 59 97];
c = [15 31 27 53 10];
int = [5];
lin_coords = intlinprog(-c, int, a.', b , [], [] , zeros(5, 1));
[funct_value, coords, num_op] = Genetic_algorithm_3(M, M_c, L, a, b, c,
   int) ;
disp(lin_coords.') %[0  1.2244         0    0.8468         0]
disp(coords.') %[0.0065   1.1907    0.0047    0.8336         0]

```

```
%3
a = [436 23 35; 58 232 223; 346 75 35; 83 12 97];
b = [2000 2000 2000];
c = [120 240 110 99];
int = [4];

lin_coords = intlinprog(-c, int, a.', b ,[], [] , zeros(3, 1));
[funct_value, coords, num_op] = Genetic_algorithm_3(M, M_c, L, a, b, c,
   int) ;
disp(lin_coords.') %[2.4825 7.9752   1.0753   1.0000]
disp(coords.') %[2.4705  7.9681   1.0807   1.0000]
```

К строкам вывода добавлены комментарии, которые показывают что они выводят. Второй и третий пример основываются на сравнении результатов с линейным программированием в MatLaB.