

Стохастические методы оптимизации

Аксенов Антон

1 Алгоритм имитации отжига

1.1 Описание алгоритма имитации отжига

Приведем описание алгоритма имитации отжига, который используется для минимизации некоторой функции $H : S \rightarrow \mathbb{R}$.

1. Фиксируется начальное значение температуры T_0 , а также счетчик $k = 0$.
2. Выбирается случайный элемент $s_0 \in S$.
3. Понижается температура $T_{k+1} = \alpha T_k, \alpha \in (0, 1)$.
4. Строится новый элемент $\tilde{s}_k = G(s_k)$, где $G : S \rightarrow S$ - фиксированная исследователем случайная функция.
5. Вычисляется $\Delta = H(\tilde{s}_k) - H(s_k)$.
6. Если $\Delta < 0$, то $s_{k+1} = \tilde{s}_k$. В противном случае с вероятностью $p = \exp(-\frac{\Delta}{T_k})$ полагается, что $s_{k+1} = \tilde{s}_k$. С вероятностью $1 - p$ $s_{k+1} = s_k$.
7. Полагается $k := k + 1$, переход к шагу 3.

1.2 Задача о расстановке ферзей

В данной главе будет решаться задача расстановки n ферзей на шахматной доске размера $n \times n$ так, чтобы они друг друга не били. Для решения задачи будет минимизироваться количество различных упорядоченных пар бьющих друг друга ферзей с использованием алгоритма имитации отжига.

В качестве кодировки расстановки ферзей будет использован одномерный массив из n элементов, каждый из которых отвечает за номер

горизонтали позиции соответствующего ферзя. Заметим, что для того, чтобы ферзи не били друг друга, необходимо, чтобы они стояли на разных вертикалях и горизонталях, поэтому искомая расстановка является перестановкой элементов множества $\{1, \dots, n\}$.

Сначала требуется написать функцию, которая будет по расстановке ферзей считать количество упорядоченных пар бьющих друг друга ферзей. Идея алгоритма подсчета в том, что мы будем проходиться по всем парам ферзей и считать эту пару 'бьющей', если модуль разности номеров строк и столбцов позиций ферзей одинаковы. Таким образом, код этой функции в MATLAB имеет следующий вид:

```
1 function [cnt] = h(q)
2 n = size(q, 2);
3 cnt = 0;
4 for i = 1:n
5     for j = i+1:n
6         if j - i == abs(q(j) - q(i))
7             cnt = cnt + 2;
8         end
9     end
10 end
11 end
```

Также требуется написать функцию, которая будет менять местами два случайных элемента массива, ее код будет иметь вид:

```
1 function [q_old] = swap(q_old)
2 n = size(q_old, 2);
3 idx = randi([1, n], 1, 2);
4 q_old([idx(1), idx(2)]) = q_old([idx(2), idx(1)]);
5 end
```

Теперь реализуем алгоритм имитации отжига в виде функции, которая на вход будет принимать три параметра: n - размер доски и одновременно количество ферзей, α - множитель, на который умножается температура, t_0 - начальная температура. В качестве выхода будет выступать итоговая расстановка ферзей q , количество шагов алгоритма k и общее количество итераций.

Основная идея алгоритма состоит в том, что мы будем менять номера горизонталей двух случайных ферзей местами - при уменьшении количества бьющих ферзей выберем новую расстановку, в противном случае выберем ее с некоторой вероятностью, которая будет зависеть от прироста целевой функции и постепенно уменьшающейся температуры. Когда

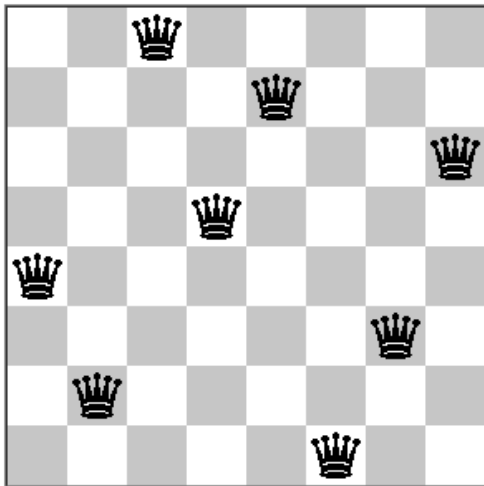
будет получена искомая расстановка, процесс остановится. Код функции в MATLAB:

```
1 function [q, k] = annealing(n, alpha, t_0)
2 t = t_0;
3 q = 1:n;
4 strikes = h(q);
5 k = 0;
6 while strikes ~= 0
7     t_new = alpha * t;
8     q_new = swap(q);
9     strikes_new = h(q_new);
10    delta_strikes = strikes_new - strikes;
11    p = exp(-delta_strikes / t);
12    xi = binornd(1, p);
13    if or(delta_strikes < 0, xi == 1)
14        q = q_new;
15        strikes = strikes_new;
16    end
17    t = t_new;
18    k = k + 1;
19 end
```

Протестируем полученную программу, для чего рассмотрим случай 8 ферзей и обычной шахматной доски 8×8 , в качестве α возьмем 0.8 (значение было подобрано на основании результатов нескольких запусков, минимизируя количество итераций алгоритма), а начальную температуру положим равной 100. Тогда получим следующее:

```
1 >> [q, k] = annealing(8, 0.8, 100)
2
3 q =
4
5 5      7      1      4      2      8      6      3
6
7
8 k =
9
10 60
11
12 >>
```

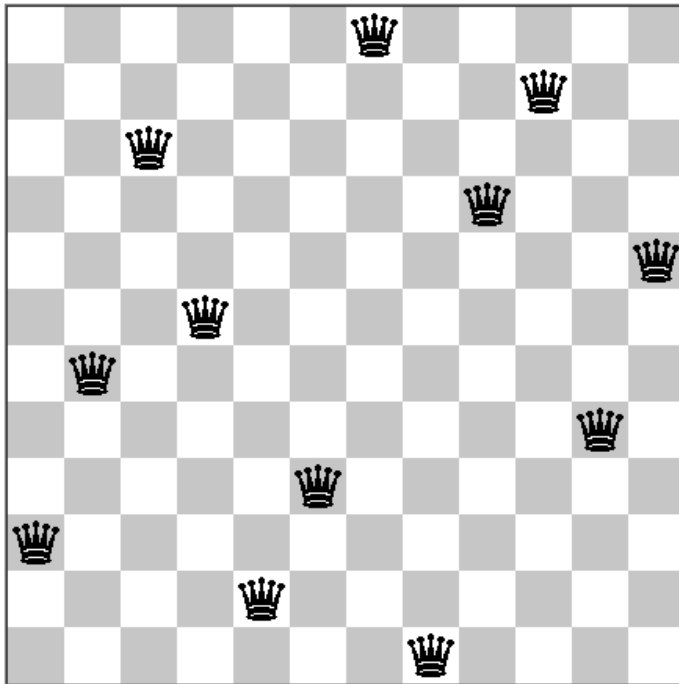
Нетрудно заметить, что в полученной расстановке ферзи не бьют друг друга, продемонстрируем это визуально.



Сделаем то же самое, но с 12 ферзями:

```
1 >> [q, k] = annealing(12, 0.8, 100)
2
3 q =
4
5 10      7      3      6      11      9      1      12      4
6          2      8      5
7
8 k =
9
10 628
11
12 >>
```

На этот раз ферзи также не бьют друг друга:



Попробуем увеличить размерность задачи и расставим 20 ферзей.

```

1 >> [q, k_0] = annealing(20, 0.8, 100);
2 [q, k_1] = annealing(20, 0.8, 100);
3 [q, k_2] = annealing(20, 0.8, 100);
4 [q, k_3] = annealing(20, 0.8, 100);
5 [q, k_4] = annealing(20, 0.8, 100);
6 [k_0, k_1, k_2, k_3, k_4]
7
8 ans =
9
10 499    654    425    467    219
11
12 >>

```

Видно, что количество шагов и итераций достаточно низкое, поэтому можно сказать, что алгоритм достаточно быстро справился с поставленной задачей.

Даже если увеличить количество ферзей до 100, то итераций будет по-прежнему немного.

```

1 >> [q, k_0] = annealing(100, 0.8, 100);
2 [q, k_1] = annealing(100, 0.8, 100);

```

```

3 [q, k_2] = annealing(100, 0.8, 100);
4 [q, k_3] = annealing(100, 0.8, 100);
5 [q, k_4] = annealing(100, 0.8, 100);
6 [k_0, k_1, k_2, k_3, k_4]
7
8 ans =
9
10 6036          3693          5115          8328          12685
11
12 >>

```

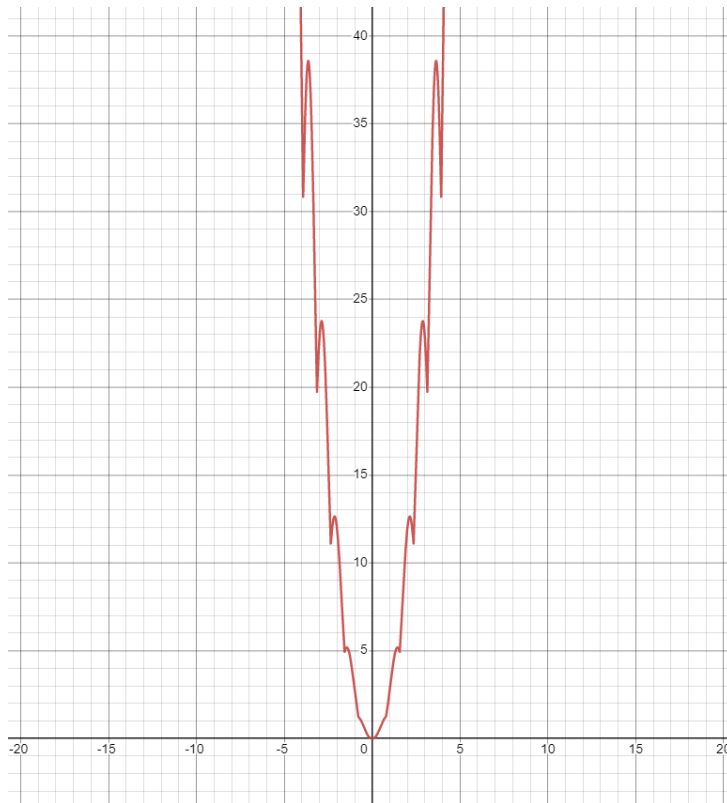
Таким образом, задача по применению метода имитации отжига к решению задачи о расстановке ферзей на шахматной доске так, чтобы они друг друга не били, успешно решена.

1.3 Минимизация вещественнозначной функции

В данной главе будет использован метод имитации отжига для минимизации следующей функции

$$f(x) = x^2(2 + |\sin 4x|), x \in \mathbb{R}$$

Построим ее график:



Очевидно, что ее глобальный минимум равен нулю, и он достигается при $x = 0$.

Теперь опишем детали применения алгоритма имитации отжига к данной задаче. Преобразование точки x для получения новой точки x' будет иметь следующий вид:

$$x' = x + \xi, \text{ где } \xi \sim U[-1, 1]$$

Начальное приближение x_0 будет браться из равномерного распределения на отрезке $[-10, 10]$, то есть

$$x_0 \sim U[-10, 10]$$

В качестве критерия останова будет выступать достижение температурой близкого к нулю значения.

Тогда код алгоритма имитации отжига будет иметь следующий вид:

```

1 function [x, y] = annealing_alg(alpha, t_0, t_thr)
2 t = t_0;
3 x = unifrnd(-10, 10);
4 y = x^2 * (2 + abs(sin(4 * x)));

```

```

5 while t > t_thr
6     x_new = x + unifrnd(-1, 1);
7     y_new = x_new^2 * (2 + abs(sin(4 * x_new)));
8     delta_y = y_new - y;
9     p = exp(-delta_y / t);
10    xi = binornd(1, p);
11    if or(delta_y < 0, xi == 1)
12        x = x_new;
13        y = y_new;
14    end
15    t = t * alpha;
16 end

```

Возьмем значения параметров, которые были использованы на лекции Шамина, то есть начальную температуру положим равной 100, нижнюю границу температуры - 10^{-10} , а также пусть $\alpha = 0.95$. Запустим несколько раз наш алгоритм:

```

1 [x, y] = annealing_alg(0.95, 100, 10^(-10))
2
3 x =
4
5 0.0022
6
7
8 y =
9
10 9.6014e-06
11
12 >> [x, y] = annealing_alg(0.95, 100, 10^(-10))
13
14 x =
15
16 -1.2132e-04
17
18
19 y =
20
21 2.9446e-08
22
23 >> [x, y] = annealing_alg(0.95, 100, 10^(-10))
24

```



```
25 | x =  
26 |  
27 | -0.0177  
28 |  
29 |  
30 | y =  
31 |  
32 | 6.4701e-04  
33 |  
34 | >>
```

Во всех тестах оптимальная точка колеблется около нуля, который является точкой глобального минимума целевой функции. Также каждая из итераций была выполнена достаточно быстро.

Таким образом, несмотря на то, что алгоритм имитации отжига и не нашел точный глобальный минимум, он подобрался к нему достаточно близко, при этом сделал это довольно быстро.

2 Метод роения частиц

В данной главе будет использован метод роения частиц для минимизации нескольких трудных для оптимизации функций.

Для начала реализуем метод роения частиц, причем для ускорения работы алгоритма постараемся избежать использования циклов там, где это возможно. Вместо этого будем стараться выполнять операции в матричном виде.

Также стоит отметить, что начальное приближение вектора положения частиц и их скоростей будет генерироваться из равномерного распределения на отрезке $[\underline{x}, \bar{x}]$ и $[\underline{x} - \bar{x}, \bar{x} - \underline{x}]$ соответственно. Параметры \underline{x}, \bar{x} будут задаваться исследователем.

В таком случае код метода роения частиц будет иметь следующий вид:

```
1 function [argvalue_global, argmin_global] =  
    particle_swarm(f, dim, x_lower, x_upper, alpha,  
        beta, gamma, M, L)  
2 X = unifrnd(x_lower, x_upper, M, dim);  
3 V = unifrnd(x_lower - x_upper, x_upper - x_lower, M,  
    dim);  
4 X = X + V;  
5 y = f(X);  
6 argvalue_local = y;  
7 argmin_local = X;  
8  
9 [argvalue_global, idxmin_global] = min(  
    argvalue_local);  
10 argmin_global = argmin_local(idxmin_global, :);  
11  
12 for t = 1:L  
13     xi_1 = repmat(unifrnd(0, 1, M, 1), 1, dim);  
14     xi_2 = repmat(unifrnd(0, 1, M, 1), 1, dim);  
15     V = alpha * V + beta * xi_1 .* (argmin_local - X)  
        + gamma * xi_2 .* (argmin_global - X);  
16 X = X + V;  
17 y = f(X);  
18 [argvalue_local, idxmin_local] = min([  
    argvalue_local, y], [], 2);  
19 mask_idx = find(idxmin_local == 2);  
20 argmin_local(mask_idx, :) = X(mask_idx, :);
```

```

21 [argvalue_global, idxmin_global] = min(
    argvalue_local);
22 argmin_global = argmin_local(idxmin_global, :);
23 end

```

Протестируем данный метод на функции Розенброка, которая имеет следующий вид:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Видно, что глобальный минимум функции достигается в точке $(x, y) = (1, 1)$

Положим $\underline{x} = -10, \bar{x} = 10, \alpha = 0.95, \beta = 0.2, \gamma = 0.2$, а также количество частиц зафиксируем на уровне 100, количество итераций будет равным 1000. В результате применения метода получим следующее:

```

1 >> F = @(X)(1 - X(:, 1)).^2 + 100 * (X(:, 2) - X(:,
    1).^2).^2
2
3 F =
4
5 function_handle with value:
6
7 @(X)(1-X(:,1)).^2+100*(X(:,2)-X(:,1).^2).^2
8
9 >> [y, x] = particle_swarm(F, 2, -10, 10, 0.95, 0.2,
    0.2, 100, 1000)
10
11 y =
12
13 5.3524e-22
14
15
16 x =
17
18 1.0000    1.0000

```

Таким образом, глобальный минимум был найден, причем алгоритм отработал очень быстро, чему поспособствовало использование матричных вычислений.

Теперь протестируем метод роя частиц на другой трудной для оптимизации функции, имеющей следующий вид:

$$f(x, y, z) = 0.01(x - 0.5)^2 + |x^2 - y| + |x^2 - z|$$

$$-0.2 \leq x \leq 1, \quad -0.3 \leq y \leq 1, \quad -0.5 \leq z \leq 1$$

Известно, что глобальный минимум этой функции достигается в точке $(x, y, z) = (0.5, 0.25, 0.25)$

Применим метод роев частиц к этой функции, пусть $\underline{x} = -1, \bar{x} = 1$, а количество итераций возьмем равным 10^6 (оно было подобрано на основании нескольких запусков). В результате получим следующее:

```

1
2 >> F = @(X)0.01 * (X(:, 1) - 0.5).^2 + abs(X(:, 1)
3      .^2 - X(:, 2)) + abs(X(:, 1).^2 - X(:, 3))
4
5 F =
6 function_handle with value:
7
8 @(X)0.01*(X(:,1)-0.5).^2+abs(X(:,1).^2-X(:,2))+abs(X
9      (:,1).^2-X(:,3))
10
11 >> [y, x] = particle_swarm(F, 3, -1, 1, 0.95, 0.2,
12      0.2, 100, 10^6)
13
14 y =
15
16
17 x =
18
19 0.5000    0.2500    0.2500
20
21 >>
```

Видно, что глобальный минимум был найден. Алгоритм работал примерно 10-12 секунд, что можно считать неплохим результатом.

Таким образом, в данной главе был реализован метод роев частиц в MATLAB и с помощью него были успешно найдены глобальные минимумы двух непростых для оптимизации функций.

3 Генетический алгоритм

В данной главе будет реализована одна вариация генетического алгоритма, которая затем будет протестирована на нескольких задачах.

3.1 Описание генетического алгоритма

Для начала кратко опишем генетический алгоритм: данная процедура используется для решения задачи минимизации некоторой функции $f : \mathbb{R}^n \rightarrow \mathbb{R}$ на некотором множестве $D \subseteq \mathbb{R}^n$. Также предполагается, что она неотрицательная, иначе говоря $f(x) \geq 0 \forall x \in D$.

В рамках алгоритма генами будут называться вектора переменных, то есть точки $x \in D$. Хромосомами будут называться компоненты данного вектора.

Идея алгоритма состоит в создании некоторой популяции генов, которая будет постепенно эволюционировать и находить все лучшее приближение минимума $f(x)$. Размер популяции M , количество итераций L и заменяемых генов M_c задается исследователем. Опишем основные шаги рассматриваемой вариации генетического алгоритма:

1. Генерация популяции генов случайным образом.
2. Подсчет функции приспособленности гена $fit(x) = \frac{1}{1+f(x)}$ и сортировка популяции по убыванию значений приспособленности.
3. Замена каждой особи с номером $m \in [2, M - M_c]$ на результат ее скрещивания со случайной особью в пуле.

Скрещивание двух генов $x, y \in \mathbb{R}^n$ подразумевает создание нового гена z , где каждая хромосома z_i будет взята случайно из множества $\{x_i, y_i\}$, причем $\mathbb{P}(z_i = x_i) = \frac{fit(x)}{fit(x) + fit(y)}$.

4. Замена M_c наименее приспособленных особей случайно сгенерированными генами.
5. Возврат к шагу 2, если количество итераций не достигло L .

Заметим, что лучшая особь на каждой итерации не изменяется, поэтому алгоритм гарантированно не ухудшает уже найденное приближение. Иначе говоря, последовательность значений функции будет монотонно(нестрого) убывать. Если $f(x)$ – ограничена на множестве D , то по теореме Вейерштрасса вышеописанная последовательность будет иметь предел.

Далее рассмотрим применение генетического алгоритма на нескольких задачах.

3.2 Минимизация числовой функции

Пусть

$$f(x) = \sum_{k=1}^n x_k^2 (1 + |\sin(100x_k)|)$$

Требуется минимизировать данную функцию на множестве $D = [-10, 10]^n$. Нетрудно видеть, что глобальным минимумом является нулевой вектор размера n . Однако градиентными методами данную задачу решить представляется очень сложным ввиду большого количества локальных минимумов. Подтверждением этого тезиса является график функции для $n = 1$.

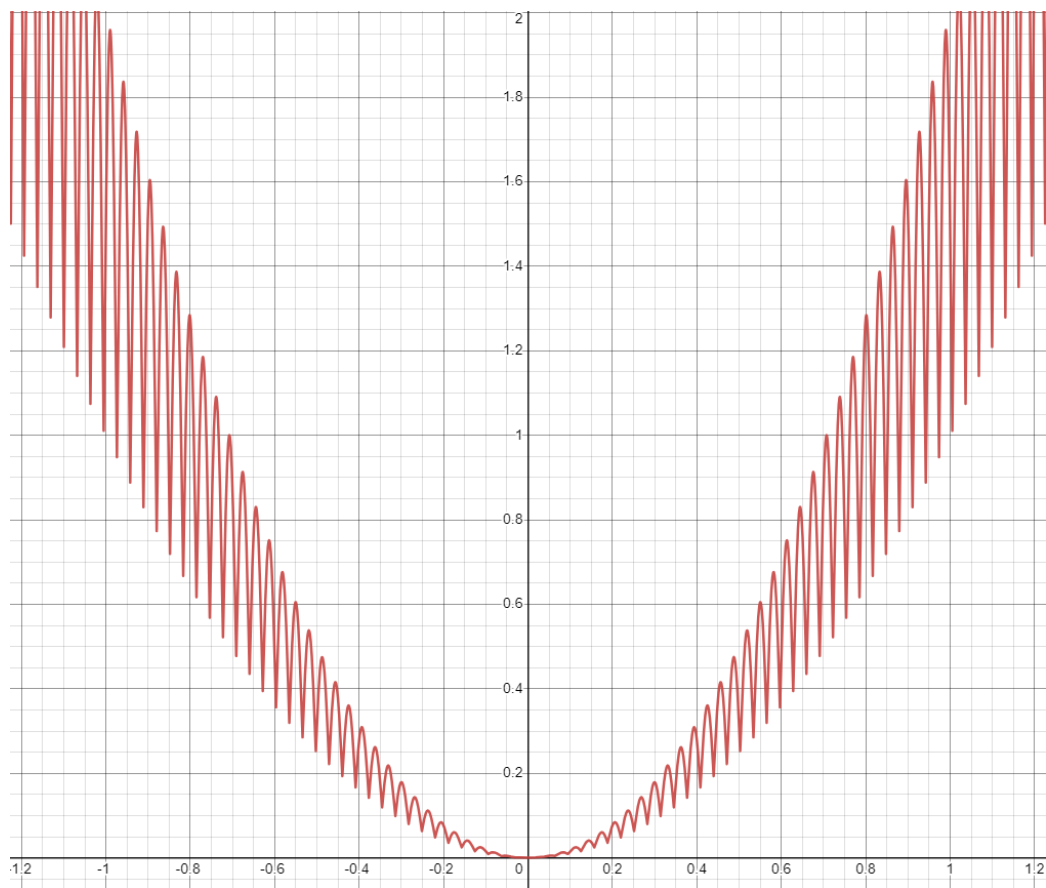


Рис. 1: График функции $f(x) = x^2(1 + |\sin(100x)|)$

Рассмотрим случай $n = 5$. Сначала реализуем расчет значений функции $f(x)$ в MATLAB:

```

1 function [y] = gen_alg_test_1(X)
2 dim = size(X, 2);
3 y = 0;
4 for i = 1:dim
5     y = y + X(:, i).^2 .* (1 + abs(sin(100 * X(:, i))))
6     );
7 end

```

Далее напишем код генетического алгоритма в соответствии с выше-описанными шагами для случая $n = 5$. Генерация гена будет происходить из равномерного распределения на множестве $[-10, 10]^5$.

```

1 function [y_min, x_min] = gen_alg_unif(f, D, M, M_c,
    L)
2 dim = size(D, 1);
3 X = zeros(M, dim);
4 for i = 1:dim
5     x_lower = D(i, 1);
6     x_upper = D(i, 2);
7     X(:, i) = unifrnd(x_lower, x_upper, M, 1);
8 end
9 for t = 1:L
10     fit = ones(M, 1) ./ (1 + f(X));
11     X = [fit X];
12     X = sortrows(X, 1, "descend");
13     fit = X(:, 1);
14     X = X(:, 2:(dim + 1));
15     male_idx = 2:M - M_c;
16     female_idx = randsample(1:length(M), M - M_c - 1,
        true);
17     male = X(male_idx, :);
18     female = X(female_idx, :);
19     child_male_gen_ind = zeros(M - M_c - 1, dim);
20     male_proba = fit(male_idx) / (fit(male_idx) + fit(
        female_idx));
21     for i = 1:M - M_c - 1
22         child_male_gen_ind(i, :) = binornd(1,
            male_proba(i), 1, dim);
23     end
24     X = male .* child_male_gen_ind + female .* (1 -
        child_male_gen_ind);

```

```

25     replace_ind = (M - M_c + 1):M;
26     for i = 1:dim
27         x_lower = D(i, 1);
28         x_upper = D(i, 2);
29         X(replace_ind, i) = unifrnd(x_lower,
30                                     x_upper, M_c, 1);
31     end
32 end
33 x_min = X(1, :);
34 y_min = f(x_min);

```

Зафиксируем параметры алгоритма: $M = 1000$, $M_c = 200$, $L = 100$. В результате запуска получим:

```

1  >> [y_min, x_min] = gen_alg_unif(@gen_alg_test_1,
2      repmat([-10, 10], 5, 1), 1000, 300, 100)
3  y_min =
4
5  0
6
7
8  x_min =
9
10 0      0      0      0      0

```

Видно, что алгоритм нашел глобальный минимум, причем время исполнения не превышает одной секунды.

3.3 Дискретная задача

Перейдем к следующей задаче – имеется множество $A = \{1, 2, \dots, n\}$, которое необходимо разбить на два непересекающихся множества K_1 и K_2 так, чтобы

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

была минимальна. Причем $K_1 \cup K_2 = A$, $K_1 \cap K_2 = \emptyset$.

Известно, что данная задача не решается точно за полиномиальное время – воспользуемся генетическим алгоритмом для нахождения приближенного решения.

Заметим, что разбиение A взаимно-однозначно задается характеристическим вектором множества K_1 , то есть вектором $x = \{\mathbb{I}\{i \in K_1\}\}_{i=1}^n$. Тогда вышеописанная задача сводится к минимизации следующей функции

$$f(x) = \left| \sum_{k=1}^n k \cdot \mathbb{I}\{x_k = 1\} - \sum_{k=1}^n k \cdot \mathbb{I}\{x_k = 0\} \right| = \left| \sum_{k=1}^n k \cdot (\mathbb{I}\{x_k = 1\} - \mathbb{I}\{x_k = 0\}) \right|$$

, где $x \in \{0, 1\}^n$

Решим эту задачу, используя MATLAB. Реализация функции $f(x)$ будет иметь вид:

```

1 function [y] = gen_alg_test_2(X)
2 dim = size(X, 2);
3 start = 1;
4 stop = start + dim - 1;
5 A = repmat(start:stop, size(X, 1), 1);
6
7 Z = zeros(size(X, 1), dim);
8 mask_0 = X == 0;
9 Z(mask_0) = A(mask_0);
10 sum_0 = sum(Z, 2);
11
12 Z = zeros(size(X, 1), dim);
13 mask_1 = X == 1;
14 Z(mask_1) = A(mask_1);
15 sum_1 = sum(Z, 2);
16
17 y = abs(sum_1 - sum_0);

```

Генетический алгоритм будет применяться аналогично предыдущей задаче, однако теперь для создания случайного гена каждая хромосома будет независимо генерироваться из распределения Бернулли с параметром $p = \frac{1}{2}$. Тогда код генетического алгоритма будет следующим:

```

1 function [y_min, x_min] = gen_alg_bern(f, dim, M,
    M_c, L)
2 X = zeros(M, dim);
3 X = binornd(1, 0.5, M, dim);
4 for t = 1:L
5     fit = ones(M, 1) ./ (1 + f(X));
6     X = [fit X];

```

```

7   X = sortrows(X, 1, "descend");
8   fit = X(:, 1);
9   X = X(:, 2:(dim + 1));
10  male_idx = 2:M - M_c;
11  female_idx = randsample(1:length(M), M - M_c - 1,
    true);
12  male = X(male_idx, :);
13  female = X(female_idx, :);
14  child_male_gen_ind = zeros(M - M_c - 1, dim);
15  male_proba = fit(male_idx) / (fit(male_idx) + fit(
    female_idx));
16  for i = 1:M - M_c - 1
17      child_male_gen_ind(i, :) = binornd(1, male_proba
    (i), 1, dim);
18  end
19  X = male .* child_male_gen_ind + female .* (1 -
    child_male_gen_ind);
20  replace_ind = (M - M_c + 1):M;
21  X(replace_ind, :) = binornd(1, 0.5, M_c, dim);
22 end
23 x_min = X(1, :);
24 y_min = f(x_min);

```

Оставим параметры алгоритма фиксированными на прежнем уровне и запустим оптимизацию для $n = 100, n = 1000, n = 10000$.

```

1  >> [y_min, ~] = gen_alg_bern(@gen_alg_test_2, 100,
    1000, 300, 100)
2
3  y_min =
4
5  0
6
7  >> [y_min, ~] = gen_alg_bern(@gen_alg_test_2, 1000,
    1000, 300, 100)
8
9  y_min =
10
11  0
12
13
14 >> [y_min, ~] = gen_alg_bern(@gen_alg_test_2, 10000,

```

```

15         1000, 300, 100)
16 y_min =
17
18 22

```

Заметим, что при $n = 100$ и $n = 1000$ генетический алгоритм нашел оптимальное разбиение исходного множества на два подмножества с одинаковой суммой. При $n = 10000$ суммы уже получились не одинаковыми, однако значение целевой функции оказалось относительно близким к нулю.

3.4 Линейное программирование

Задача линейного программирования заключается в минимизации линейной функции при условии ограничений-равенств и ограничений-неравенств, где функции ограничения имеют линейный вид. Кроме того, на некоторые переменные может быть наложено ограничение целочисленности и нахождение на некотором отрезке. Таким образом, данная задача будет иметь следующий вид

$$\begin{cases} f(x) = c^T x \rightarrow \min_{x \in \mathbb{R}^n} \\ Ax \leq b \\ A_{eq}x = b_{eq} \\ lb_i \leq x_i \leq ub_i \quad \forall i \in J \\ x_i \in \mathbb{Z} \quad \forall i \in Z \end{cases}$$

, где $A, A_{eq} \in Mat_{m \times n}(\mathbb{R})$; $b, b_{eq} \in \mathbb{R}^m$; $c \in \mathbb{R}^n$; $J, Z \subseteq \{1, 2, \dots, n\}$; $lb_i, ub_i \in \mathbb{R} \quad \forall i \in J$

Теперь от данной общей постановки перейдем к более узкой, где нет ограничений-равенств, ограничений на целочисленность, верхних границ значений переменных. Также будем рассматривать только неотрицательные значения переменных и параметров задачи. Кроме того, теперь задача будет заключаться в максимизации целевой функции. То есть

$$\begin{cases} f(x) = c^T x \rightarrow \max_{x \in \mathbb{R}^n} \\ Ax \leq b \\ x_i \geq 0 \quad \forall i \in \{1, 2, \dots, n\} \\ x_i \in \mathbb{Z} \quad \forall i \in Z \end{cases}$$

, где $A \in Mat_{m \times n}(\mathbb{R}_+)$; $b \in \mathbb{R}_+^m$; $c \in \mathbb{R}_+^n$; $Z \subseteq \{1, 2, \dots, n\}$;

Данная задача чаще всего решается симплекс-методом, однако генетический алгоритм к ней так же применим – реализуем его в MATLAB.

Заметим, что основная часть этого алгоритма уже была реализована в вышеописанных задачах, поэтому необходимо лишь адаптировать эти наработки под задачу линейного программирования. В частности, необходимо поменять генерацию генов, а также подсчет функции приспособленности.

В силу того что все переменные и параметры неотрицательны, для каждой переменной может быть вычислен отрезок, в который она точно должна попасть в соответствии с ограничениями неравенствами. Легко видеть, что

$$x_j \in [0, v_j], \text{ где } v_j = \min_{i: x_{ij} \neq 0} \frac{b_i}{x_{ij}}$$

Тогда каждую вещественнозначную хромосому для j -ой компоненты гена можно генерировать из равномерного распределения на отрезке $[0, v_j]$, а каждую целочисленную - на отрезке $[0, \lfloor v_j \rfloor]$. Если правая граница равна нулю, то соответствующая переменная всегда будет равна нулю. Если правая граница оказалась меньше нуля, то исходная задача не имеет решений.

Однако стоит отметить, что вышеописанное условие является лишь необходимым, но не достаточным – оно гарантирует неотрицательность всех переменных и целочисленность соответствующих компонент гена, однако условие $Ax \leq b$ нужно проверять дополнительно. Чтобы учесть это в рамках генетического алгоритма, присвоим нулевые значения приспособленности тем генам, для которых не выполняется $Ax \leq b$. Для остальных особей приспособленность будет считаться как

$$fit(x) = c^T x$$

Тогда код генетического алгоритма будет иметь следующий вид

```

1 function [y_best, x_best] = gen_alg_lp(A, c, b,
    int_idx, M, M_c, L)
2 % get continuous variables indexes
3 n = size(A, 2);
4 cont_idx = 1:n;
5 cont_idx(int_idx) = [];
6 % determine bounds of variables
7 B = repmat(b, 1, n);
8 x_max = min(B ./ (A + 1e-8))';
9 x_min = zeros(n, 1);

```

```

10
11 % generate population
12 X = zeros(M, n);
13 % continious variables
14 for i = cont_idx
15     x_lower = x_min(i);
16     x_upper = x_max(i);
17     X(:, i) = unifrnd(x_lower, x_upper, M, 1);
18 end
19 % integer variables
20 for i = int_idx
21     x_lower = x_min(i);
22     x_upper = floor(x_max(i)) + 1;
23     X(:, i) = randi([x_lower, x_upper], M, 1);
24 end
25
26 % main loop
27 for t = 1:L
28     % profit calc
29     y = X * c;
30     % fit func calc and checking constraint
31     res_usage = A * X';
32     res_cond_mask = all(res_usage <= repmat(b, 1, M),
33         1);
34     fit = y;
35     fit(~res_cond_mask) = 0;
36     % sort gens
37     X = [fit X];
38     X = sortrows(X, 1, "descend");
39     fit = X(:, 1);
40     X = X(:, 2:(n + 1));
41     % crossing
42     male_idx = 2:M - M_c;
43     female_idx = randsample(1:length(M), M - M_c - 1,
44         true);
45     male = X(male_idx, :);
46     female = X(female_idx, :);
47     child_male_gen_ind = zeros(M - M_c - 1, n);
48     male_proba = fit(male_idx) ./ (fit(male_idx) + fit
49         (female_idx) + 1e-8);
50     for i = 1:M - M_c - 1

```

```

48     child_male_gen_ind(i, :) = binornd(1, male_proba
49         (i), 1, n);
50 end
51 X = male .* child_male_gen_ind + female .* (1 -
52     child_male_gen_ind);
53 replace_ind = (M - M_c + 1):M;
54 % replacing M_c gens
55 % cont variables
56 for i = cont_idx
57     x_lower = x_min(i);
58     x_upper = x_max(i);
59     X(replace_ind, i) = unifrnd(x_lower, x_upper,
60         M_c, 1);
61 end
62 % integer variables
63 for i = int_idx
64     x_lower = x_min(i);
65     x_upper = floor(x_max(i)) + 1;
66     X(replace_ind, i) = randi([x_lower, x_upper],
67         M_c, 1);
68 end
69 end
70 x_best = X(1, :);
71 y_best = x_best * c;

```

Теперь протестируем данный алгоритм на конкретной задаче и сравним полученный результат с симплекс-методом.

Пусть

$$A = \begin{pmatrix} 10 & 20 & 30 \\ 60 & 20 & 10 \\ 35.4 & 40.2 & 50 \end{pmatrix} \quad b = \begin{pmatrix} 1000 \\ 750 \\ 830.5 \end{pmatrix} \quad c = \begin{pmatrix} 30 \\ 15 \\ 22.5 \end{pmatrix} \quad x_2, x_3 \in \mathbb{Z}$$

Параметры алгоритма возьмем следующими: $M = 1000$, $M_c = 200$, $L = 300$. Тогда выполним максимизацию исходной функции с помощью генетического алгоритма и встроенного в MATLAB симплекс-метода.

```

1 >> M = 1000;
2 M_c = 200;
3 L = 300;
4 >> A = [[10, 20, 30]; [60, 20, 10]; [35.4, 40.2,
5     50]];

```

```

5 | b = [1000, 750, 830.5]';
6 | c = [30, 15, 22.5]';
7 | int_idx = [2, 3];
8 | >> simp_meth = intlinprog(-c, int_idx, A, b ,[], []
   | , zeros(3, 1))
9 | LP:                               Optimal objective value is
   | -528.968254.
10 |
11 | Heuristics:                       Found 2 solutions using ZI round.
12 | Upper bound is -511.009070.
13 | Relative gap is 2.72%.
14 |
15 | Cut Generation:                   Applied 1 mir cut.
16 | Lower bound is -524.957627.
17 | Relative gap is 0.00%.
18 |
19 |
20 | Optimal solution found.
21 |
22 | Intlinprog stopped at the root node because the
   | objective value is within a gap tolerance of the
   | optimal value,
23 | options.AbsoluteGapTolerance = 0 (the default value)
   | . The intcon variables are integer within
   | tolerance,
24 | options.IntegerTolerance = 1e-05 (the default value)
   | .
25 |
26 |
27 | simp_meth =
28 |
29 | 10.7486
30 | 0
31 | 9.0000
32 |
33 | >> [~, x_best] = gen_alg_lp(A, c, b, int_idx, M, M_c
   | , L)
34 |
35 | x_best =
36 |
37 | 10.7450          0          9.0000

```

Видно, что генетический алгоритм нашел очень точное приближение решения исходной задачи условной оптимизации, причем его время работы не превышает 1-2 секунд.