Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Институт Компьютерных технологий и защиты информации

Кафедра Динамики процессов и управления

**ОТЧЕТ**

**по лабораторным работам**

Направление подготовки/специальность: 09.03.03 «Прикладная информатика»

Выполнил:

Шарипов Р.Р. 4316

Проверил:

Семенов П.К.

**Казань 2023**

**Пошаговый метод**

Метод пошагового изменения шага является численным методом решения задачи поиска минимума (или максимума) функции. Он относится к классу методов оптимизации и широко используется для приближенного нахождения экстремумов функций.

**Итерационная формула:**

Итерационная формула обычно выглядит следующим образом: *xn+1*​=*g*(*xn*) где *g*(*x*) - функция, которая определяет, как вычисляется новое приближенное значение на основе текущего.

**Условия сходимости:**

Метод может сходиться к локальному экстремуму, если выполнены условия:

* Math.Abs(h) > eps
* Начальное приближение должно быть достаточно близким к истинному значению минимума (или максимума).

**public static double StepByStep(double xn, double h, double eps, Function func)**

Входные параметры:

* **xn**: Начальное значение переменной, с которого начинается поиск.
* **h**: Начальный шаг, используемый при итерациях.
* **eps**: Заданная точность, при достижении которой поиск завершается.
* **func**: Функция, для которой мы ищем минимум.

Шаги метода:

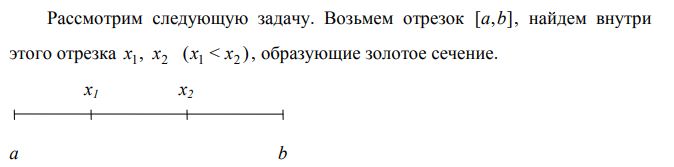
* Инициализация начальных значений: **xt** устанавливается равным начальному значению **xn**, и **ft** равно значению функции в точке **xt**.
* Итерационный процесс: Пока текущий шаг **h** больше заданной точности **eps**, выполняются следующие шаги:
* Вычисление значения функции в новой точке **xs = xt + h**.
* Сравнение значений функции в текущей и новой точках (**ft** и **fs**).
* Если значение функции в новой точке меньше, чем в текущей, увеличивается шаг (**h \*= 1.2**).
* В противном случае, шаг уменьшается вдвое (**h = -h / 2**).
* Обновление текущей точки (**xt = xs**) и значения функции в текущей точке (**ft = fs**).
* Возврат результата: Когда значение шага становится меньше или равно заданной точности, возвращается значение **xt** как приближенный минимум функции.

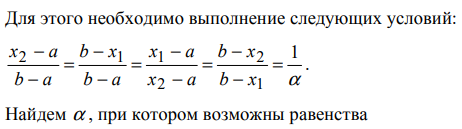
**Метод золотого сечения**

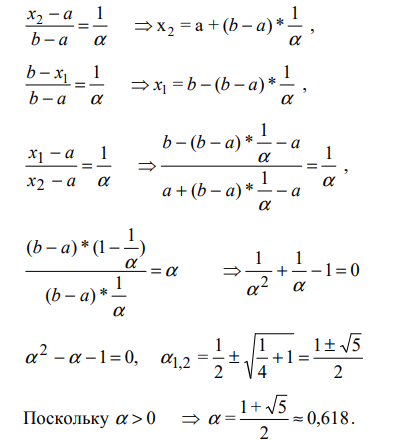
Метод золотого сечения используется для численного поиска минимума функции одной переменной. Метод золотого сечения также является последовательным методом минимизации. Опираясь на свойства золотого сечения отрезка, этот метод использует найденные значения F(х) более рационально.

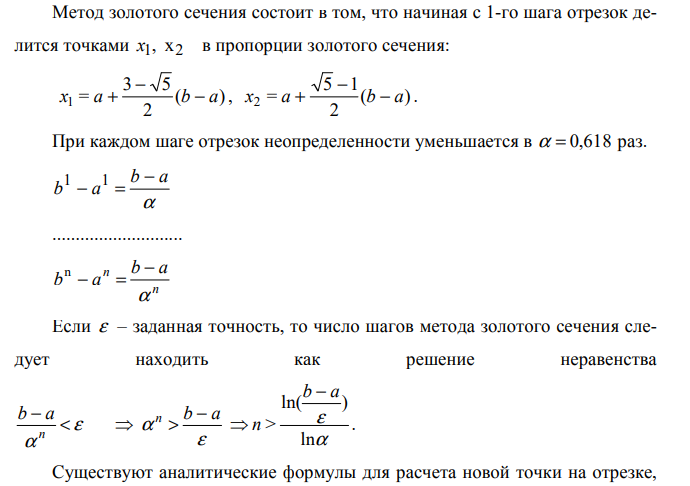
Метод основан на делении текущего отрезка [a; b], где содержится искомый экстремум, на две неравные части, подчиняющиеся правилу золотого сечения, для определения следующего отрезка, содержащего максимум.

***Правило золотого сечения:*** *отношение всего отрезка к большей его части равно отношению большей части отрезка к меньшей.*



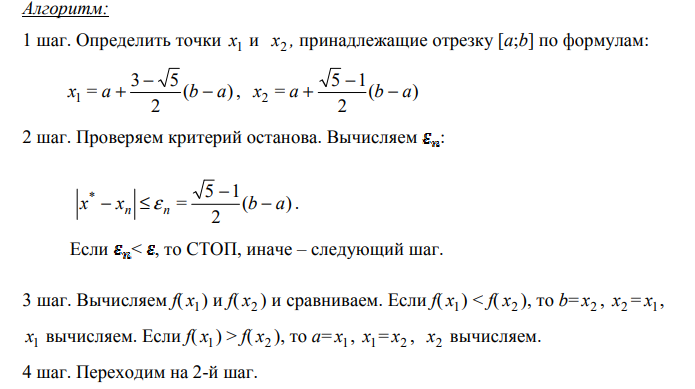


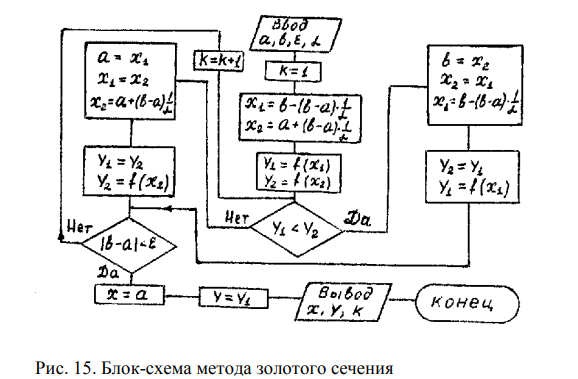






При *n* шагах метода золотого сечения *f (x)* вычисляется *n +1* раз, так как на 1-м шаге *f (x)* вычисляется дважды, а на последующих шагах по одному разу, при этом одна из внутренних точек отрезка неопределенности последующего шага совпадает с одной из точек предыдущего шага. Метод золотого сечения обеспечивает более быструю сходимость к решению, чем многие другие методы, и применим, очевидно, только для одноэкстремальных функций, т. е. функций, содержащих один экстремум того типа, который ищется в задаче.





**public static double Gold(double a, double b, double eps, Function f)**

Входные параметры:

* **a**: Левая граница начального интервала.
* **b**: Правая граница начального интервала.
* **eps**: Заданная точность, при достижении которой поиск завершается.
* **f**: Функция, для которой мы ищем минимум.

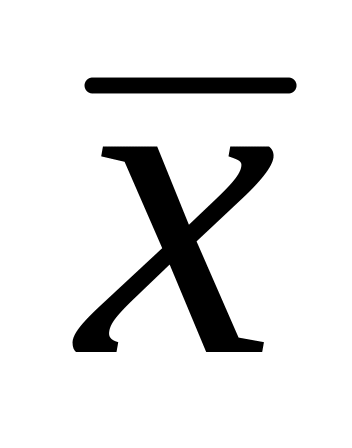
Шаги метода:

* **Инициализация границ интервала:**
* Устанавливаются начальные значения **v** и **w** внутри интервала **[a, b]**.
* Вычисляются значения функции в точках **v** и **w**: **fv** и **fw**.
* **Итерационный процесс:**
* Пока длина текущего интервала больше заданной точности **eps**, выполняются следующие шаги:
* Сравнение значений функции в точках **v** и **w** (**fv** и **fw**).
* Если **fw > fv**, обрезается правая часть интервала (**b** становится равным **w**, **w** становится **v**).
* В противном случае, обрезается левая часть интервала (**a** становится равным **v**, **v** становится **w**).
* Обновление значений **v** и **w** в соответствии с новым интервалом.
* Повторение шага до достижения заданной точности.
* **Возврат результата:**
* Когда длина интервала становится меньше или равной заданной точности, возвращается середина текущего интервала в качестве приближенного минимума.

**Метод квадратичной аппроксимации**

Метод квадратичной аппроксимации (Quadratic Approximation) используется для численного поиска минимума (или максимума) функции одной переменной.

Метод основан на предположении о том, что в ограниченном интервале можно аппроксимировать функцию квадратичным полиномом, который используется для оценивания координаты оптимума. Оценка оптимального значения рассчитывается по формуле:

= (*x*2+ *x*1)/2 - (*a*1/2*a*2).

Предполагается, что заданы *x*1, *x*2, *x*3, и известны значения функции в этих точках *f*1, *f*2, *f*3, а аппроксимирующая функция:

*g*(*x*) = *a*0+ *a*1(*x*- *x*1) + *a*2(*x*- *x*1)(*x*- *x*2)

совпадает с *f*(*x*) в трех указанных точках.

Коэффициенты полинома определяются уравнениями:

*a*0= *f*1; *a*1= (*f*2- *f*1)/(*x*2- *x*1); *a*2= 1/(*x*3- *x*2)⋅[(*f*3- *f*1)/(*x*3- *x*1) - (*f*2- *f*1)(*x*2- *x*1)].

Для унимодальных функций https://studfile.net/html/2706/633/html_bQeq0B5Nmt.hoJr/htmlconvd-v_tiwU_html_7a5632e39f74b834.gif оказывается приемлемой для оценки оптимума *x*\*.

*Алгоритм метода квадратичной аппроксимации.*

*Шаг 1*. Задать *x*1, *x*2, *x*3, и вычислить значения функции в этих точках *f(х*1), *f(х*2), *f(х*3).

*Шаг 2*. Рассчитать *a*0= *f(х*1); *a*1= (*f(х*2)- *f(х*1))/(*x*2- *x*1); *a*2= 1/(*x*3- *x*2)⋅[(*f(х*3)- *f(х*1))/(*x*3- *x*1) - (*f(х*2)- *f(х*1))(*x*2- *x*1)].

*Шаг 3*. Вычислить оптимальное решение: = (*x*2+ *x*1)/2 - (*a*1/2*a*2).

**public static double QuadraticApproximation(double xn, double h, double eps, Function f)**

Входные параметры:

* **xn**: Начальное значение переменной, с которого начинается поиск.
* **h**: Шаг для построения трех точек вокруг **xn**.
* **eps**: Заданная точность, при достижении которой поиск завершается.
* **f**: Функция, для которой мы ищем минимум.

Шаги метода:

* **Инициализация точек:**
* Выбираются три точки вокруг текущего значения **xn**: **x1 = xn - h**, **x2 = xn**, **x3 = xn + h**.
* Вычисляются значения функции в этих точках: **y1 = f(x1)**, **y2 = f(x2)**, **y3 = f(x3)**.
* **Итерационный процесс:**
* Пока разница между последним найденным минимумом и текущим значением **xs** больше заданной точности **eps**, выполняются следующие шаги:
* Вычисление коэффициентов квадратичной аппроксимации: **a**, **b**, **c**.
* Нахождение значения **xs** как точки минимума квадратичной функции.
* Оценка значений функции в точке **xs**: **fxs = f(xs)**.

Если коэффициент **a** (коэффициент при квадратичном члене) положителен:

* Обновление точек и их значений в соответствии с новым значением **xs**.
* Выбор точки с максимальным значением функции для обновления.

Если коэффициент **a** отрицателен:

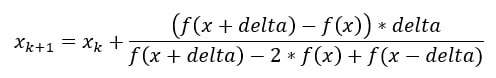
* Обновление точек и их значений в соответствии с новым значением **xs**.
* Выбор точки с минимальным значением функции для обновления.
* **Возврат результата:**
* Когда разница между последним найденным минимумом и текущим значением **xs** становится меньше или равной заданной точности, возвращается значение **xs** как приближенный минимум.

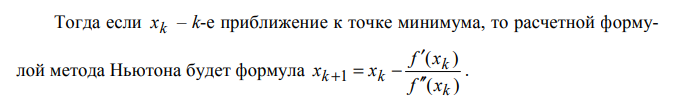
**Метод ньютона**

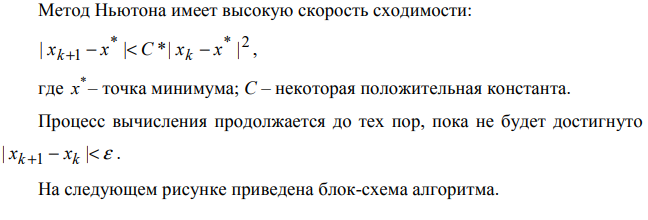
Метод Ньютона относится к методам 2-го порядка и рекомендуется к применению в том случае, когда задача минимизации достаточно хорошо локализована.

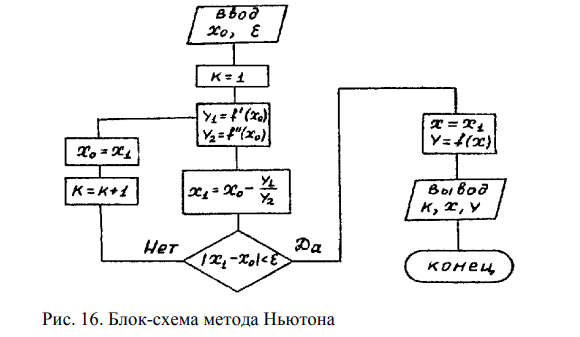
Обычно это имеет место в том случае, когда на начальном этапе применяется один из методов *0-го* порядка, а затем осуществляется переход к методу Ньютона. Для этого необходимым условием является гладкость *f (x)*, существование не равных нулю *f ′(x) f ′′(x) f ′′′(x)* для *x ∈[a,b].*

*delta = eps / 2*









public static double NewtonsMethod(double currentPoint, double eps, Function func)

Входные параметры:

* **currentPoint**: Начальная точка, с которой начинается процесс итераций.
* **eps**: Заданная точность, при достижении которой поиск завершается.
* **func**: Функция, для которой мы ищем корень или минимум.

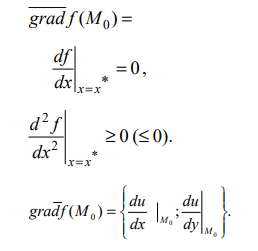
Шаги метода:

* **Инициализация переменных:**
* **delta** и **old\_delta** инициализируются для отслеживания изменения значения переменной между итерациями.
* **Итерационный процесс:**
* Повторение итераций до тех пор, пока абсолютная разность между текущей и предыдущей точкой больше заданной точности **eps**.
* На каждой итерации выполняются следующие шаги:
* Вычисление первой производной (производной первого порядка) в текущей точке.
* Вычисление второй производной (производной второго порядка) в текущей точке.
* Вычисление нового значения точки по формуле Ньютона: **newPoint = currentPoint - (derivative / secondDerivative)**.
* Расчет абсолютной разности **delta = Math.Abs(newPoint - currentPoint)**.
* Обновление текущей точки **currentPoint** значением **newPoint**.
* **Проверка на расходимость:**
* Проверка, не приводит ли текущая итерация к расходимости (если новое значение **delta** больше предыдущего).
* **Возврат результата:**
* Когда абсолютная разность **delta** становится меньше или равной заданной точности **eps**, возвращается текущая точка как приближенное решение.

**Градиентный спуск.**

Метод градиентного спуска — это алгоритм оптимизации, который используется для минимизации функции

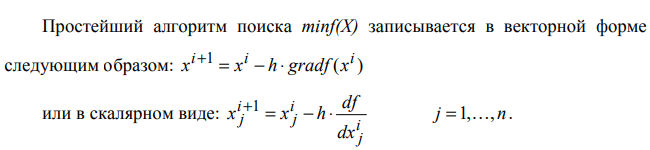
**Определение.** *Градиентом функции u=f(x,y)* в точке *М0(х0,у0)* называется вектор, координаты которого равны значениям частных производных функции в точке *М0*:

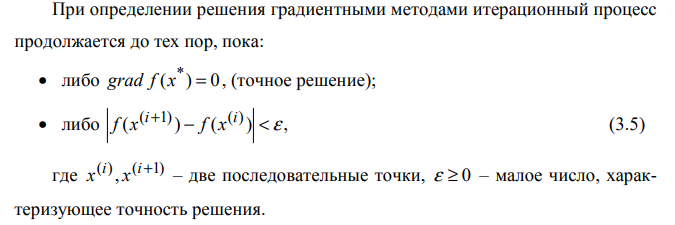


Градиент функции направлен по нормали к поверхности уровня, т. е. перпендикулярно к касательной плоскости, проведенной в точке *М* в сторону наибольшего возрастания функции в данной точке.

Градиентные методы представляют собой приближенные (итерационные) методы решения задачи нелинейного программирования и позволяют решить практически любую задачу. Однако при этом определяется локальный экстремум. Поэтому целесообразно применять эти методы для решения задач выпуклого программирования, в которых каждый локальный экстремум является и глобальным.

Основным понятием, используемым во всех градиентных методах, является понятие градиента функции, как направления наискорейшего возрастания функции. Известно, что функция многих переменных *f(X)* наиболее сильно возрастает в направлении градиента *grad f(X)* , а убывает в направлении антиградиента функции в этой точке, то есть в направлении вектора − *grad f (X)* .

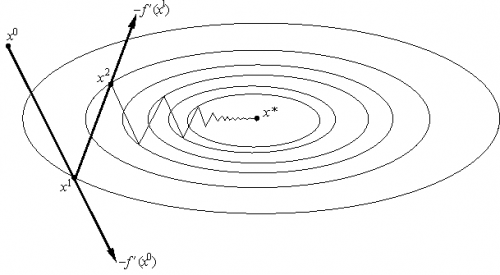


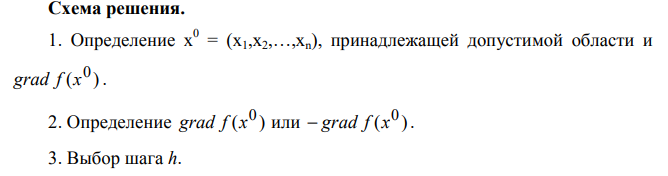


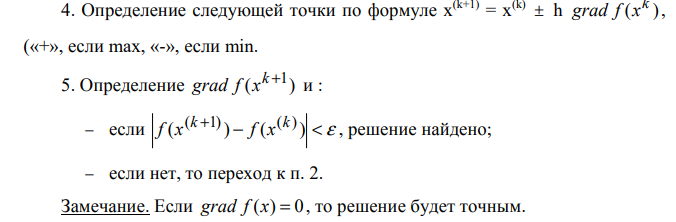
Представим человека, стоящего на склоне оврага, которому необходимо спуститься вниз (на дно). Наиболее естественным, кажется, направление в сторону наибольшей крутизны спуска, т.е. направление *(– grad f (X))*. Получаемая при этом стратегия, называемая **градиентным методом**, представляет собой последовательность шагов, каждый из которых содержит две операции:

а) определение направления наибольшей крутизны спуска (подъема);

б) перемещение в выбранном направлении на некоторый шаг.







public static OptimalResult GradientDescent(Vector startPoint, double h, double eps, Func<Vector, double> function)

**Входные параметры:**

* **startPoint**: Начальная точка (вектор), с которой начинается оптимизация.
* **h**: Начальный размер шага для градиентного спуска.
* **eps**: Порог сходимости. Алгоритм останавливается, когда изменение значения функции становится менее **eps**.
* **function**: Целевая функция, которую необходимо минимизировать.

**Обзор Реализации:**

* Инициализация текущей точки как начальной точки.
* Внутри основного цикла вычисляется текущее значение целевой функции в текущей точке.
* Инициализация новой точки для следующей итерации.
* Вычисление частных производных для каждой переменной и обновление новой точки соответственно.
* Проверка, приводит ли новая точка к более низкому значению функции. Регулировка размера шага (**h**) в зависимости от этого.
* Обновление текущей точки, если новая точка приводит к более низкому значению функции.
* Повторение процесса до сходимости или достижения максимального числа итераций.

**Методы:**

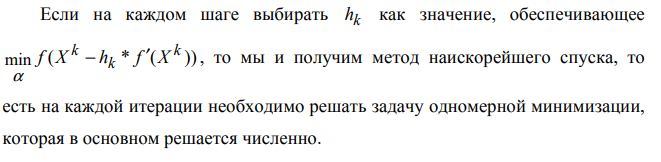
* **GetDerivative:**
* Назначение: Вычисляет численную производную заданной функции в конкретной точке.
* Параметры:
* **function**: Функция, для которой вычисляется производная.
* **point**: Точка, в которой вычисляется производная.
* **delta**: Малое возмущение для численного дифференцирования.
* Возвращает: Численная производная в указанной точке.
* **Replace:**
* Назначение: Создает новый вектор, заменяя значение в конкретном индексе входного вектора.
* Параметры:
* **point**: Исходный вектор.
* **replace**: Новое значение для замены в указанном индексе.
* **replaceIndex**: Индекс, на котором должна произойти замена.
* Возвращает: Новый вектор с указанной заменой.

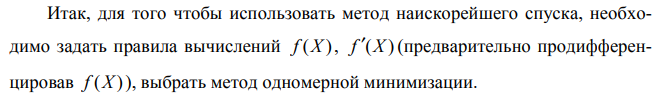
**Критерий Сходимости:** Алгоритм завершает работу, когда изменение значения функции между последовательными итерациями меньше или равно заданному порогу сходимости (**eps**).

**Регулировка Размера Шага:** Если новая точка приводит к более высокому значению функции, размер шага (**h**) уменьшается на 20%. Это делается для улучшения сходимости при перепрыгивании минимума.

**Метод наискорейшего спуска**

В отличие от метода градиента, в котором градиент определяют на каждом шаге, в методе наискорейшего спуска градиент находят в начальной точке и движение в найденном направлении продолжают одинаковыми шагами до тех пор, пока значение функции уменьшается (увеличивается). Если на каком-либо шаге *f(X)* возросло (уменьшилось), то движение в данном направлении прекращается, последний шаг снимается полностью или наполовину и вычисляется новое значение градиента и новое направление.





Этот вариант градиентного метода основывается на выборе шага из следующего соображения. Из точки x^{[k]} будем двигаться в направлении антиградиента до тех пор пока не достигнем минимума функции f на этом направлении, т. е. на луче L=\{x=x^{[k]}-\lambda f'(x^{[k]});\; \lambda \leq 0} :

\lambda^{[k]} = \arg\min_{ \lambda\in [0, \infty)} f(x^{[k]}-\lambda f'(x^{[k]})).

Другими словами, \lambda^{[k]} выбирается так, чтобы следующая итерация была точкой минимума функции f на луче L (см. рис. 3). Такой вариант градиентного метода называется методом наискорейшего спуска. Заметим, кстати, что в этом методе направления соседних шагов ортогональны.

Метод наискорейшего спуска требует решения на каждом шаге задачи одномерной оптимизации. Практика показывает, что этот метод часто требует меньшего числа операций, чем градиентный метод с постоянным шагом.

В общей ситуации, тем не менее, теоретическая скорость сходимости метода наискорейшего спуска не выше скорости сходимости градиентного метода с постоянным (оптимальным) шагом.

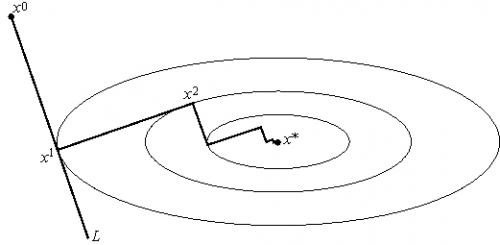


Рис.3 Геометрическая интерпретация метода наискорейшего спуска. На каждом шаге \lambda^{[k]} выбирается так, чтобы следующая итерация была точкой минимума функции f на луче L.

**public static OptimalResult SteepestDescentMethod(Vector xn, double h, double eps, Func<Vector, double> func)**

Входные параметры:

* **xn**: Начальная точка, с которой начинается процесс оптимизации.
* **h**: Начальный шаг для обновления точек (может быть скорректирован в процессе).
* **eps**: Заданная точность, при достижении которой оптимизация завершается.
* **func**: Многомерная функция, которую мы оптимизируем.

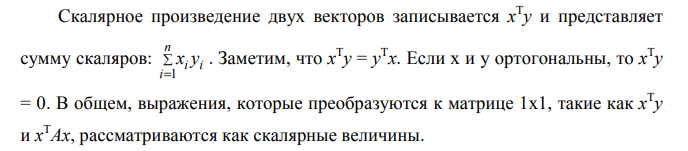
Шаги метода:

* **Инициализация переменных:**
* Инициализация начальной точки **xn**.
* Инициализация счетчика итераций **iterationCount** и предыдущего значения функции **previousObjectiveValue**.
* **Итерационный процесс:**
* Повторение итераций до тех пор, пока шаг **h** больше заданной точности **eps**.
* На каждой итерации выполняются следующие шаги:
* Вычисление градиента функции в текущей точке **xn**.
* Определение оптимального размера шага с использованием метода **StepByStep**.
* Обновление текущей точки **xs** по формуле **xs = xn - (optimalStepSize \* gradient)**.
* Обновление значения функции в новой точке **currentObjectiveValue**.
* Вычисление вектора смещения **displacement** между текущей и предыдущей точками.
* Проверка условия сходимости: если норма вектора смещения меньше заданной точности **eps**, завершение.
* **Возврат результата:**
* Возврат результатов оптимизации в виде объекта **OptimalResult**, который содержит:
* **xs**: конечная точка оптимизации.
* **gradient**: градиент в конечной точке (в данном коде не вычисляется, указано как **null**).
* **finalValue**: значение функции в конечной точке.
* **iterationCount**: количество выполненных итераций.

**Метод сопряжённых градиентов**

Название метода отражает тот факт, что данный метод отыскания безусловного экстремума сочетает в себе понятия градиента целевой функции и сопряженных направлений.

Обозначения, используемые далее.



Квадратичная форма – это просто скаляр, квадратичная функция некого вектора x следующего вида: *f(x) = (1/2)⋅xT Ax-b T x+c*

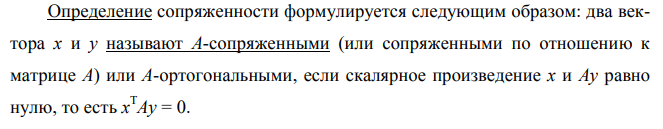
**Определение.** Матрица А называется положительно-определенной, если для любого ненулевого вектора x справедливо следующее: *xTAx > 0* (3.7). ***Матрица А называется положительно-полуопределенной***, если для любого ненулевого вектора x справедливо следующее*: xTAx ≥ 0*.

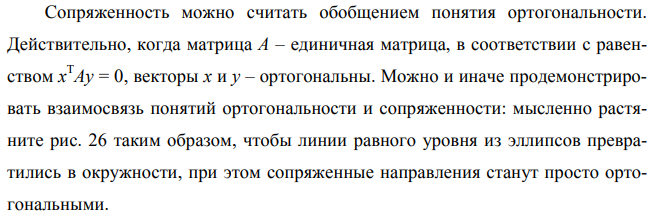
***Матрица А называется отрицательно-определенной***, если для любого ненулевого вектора x справедливо следующее: *xTAx <0*.

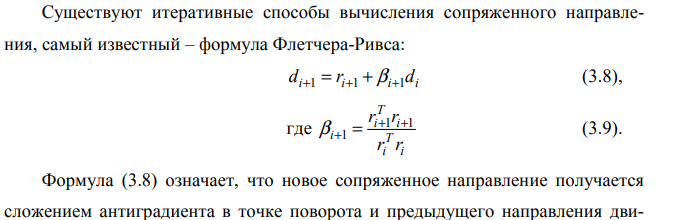
***Матрица А называется отрицательно-полуопределенной***, если для любого ненулевого вектора x справедливо следующее: *xTAx ≤ 0*.

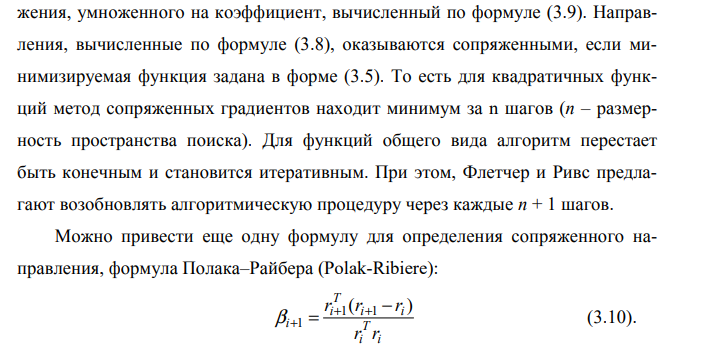
***Матрица А называется неопределенной***, если существуют ненулевые векторы x, для которых выполняются неравенства: *x T Ax <0* и *x T Ax> 0.*

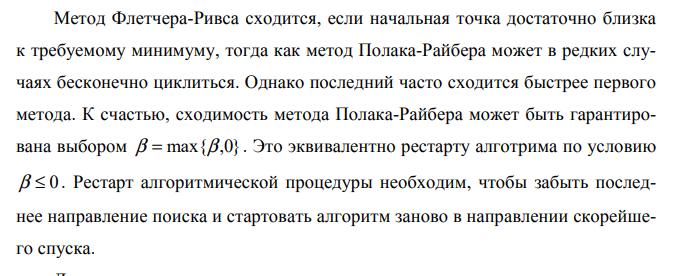
Если матрица А – положительно-определенная, можно найти минимум ее квадратичной функции. Причем, метод сопряженных градиентов сделает это за n или менее шагов, где n – размерность неизвестного вектора x

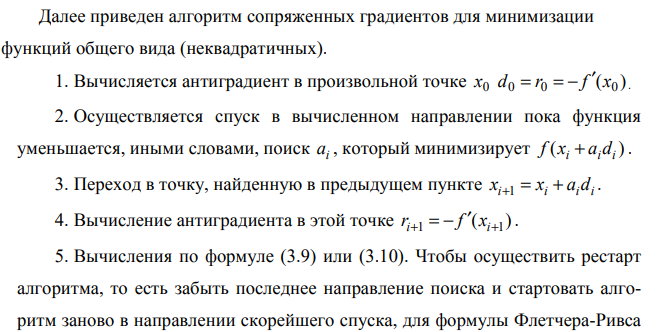


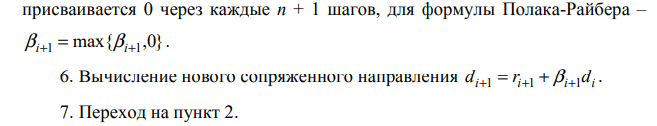














public static OptimalResult ConjugateGradientMethod(Vector initialVector, double h, double eps, Func<Vector, double> func)

Входные параметры:

* **initialVector**: Начальная точка, с которой начинается процесс оптимизации.
* **h**: Начальный шаг для обновления точек (может быть скорректирован в процессе).
* **eps**: Заданная точность, при достижении которой оптимизация завершается.
* **func**: Многомерная функция, которую мы оптимизируем.

Шаги метода:

* **Инициализация переменных:**
* Инициализация начальной точки **initialVector**.
* Инициализация счетчика итераций **iterationCount**.
* Инициализация значений текущего и предыдущего значения функции **currentObjectiveValue** и **previousObjectiveValue**.
* Инициализация градиента, следующего градиента и вектора направления поиска.
* **Итерационный процесс:**
* Повторение итераций до тех пор, пока шаг **h** больше заданной точности **eps**.
* На каждой итерации выполняются следующие шаги:
* Вычисление градиента функции в текущей точке **vector**.
* Выбор направления поиска.
* Вычисление следующего градиента.
* Обновление направления поиска с использованием формулы сопряженных градиентов.
* Оптимизация шага по направлению поиска.
* Обновление текущей точки **currentSolution**.
* Проверка сходимости: если норма разности текущей и предыдущей точек меньше заданной точности **eps**, завершение.
* **Возврат результата:**
* Возврат результатов оптимизации в виде объекта **OptimalResult**, который содержит:
* **currentSolution**: конечная точка оптимизации.
* **gradient**: градиент в конечной точке (в данном коде не вычисляется, указано как **null**).
* **finalValue**: значение функции в конечной точке.
* **iterationCount**: количество выполненных итераций.

**Метод случайного поиска**

Метод случайного поиска - это метод оптимизации, который используется для нахождения наиболее оптимального решения в заданном пространстве параметров. Он основан на выборе случайных комбинаций параметров и оценке их эффективности

Выбирают начальную точку и размер случайного вектора ***^*** такого, что ***\% = рслп –*** Так как ***рслп*** должен быть безразмерной величиной, то сначала переходят к нормированному факторному пространству, причем за нормированные единицы варьирования принимают некоторые условные интервалы варьирования ***Axt*** по каждому фактору (/= 1, 2, ..., п). За такие условные интервалы варьирования могут быть приняты, например, абсолютные погрешности измерения ***Sxt*** по каждому ***i-***му фактору или более крупные отрезки.

Определяют все п составляющих случайного вектора ***%***, началом которого служит нулевая точка (на рис. 9.10 точка А0), а конец вектора ***%*** равновероятно распределен по окружности (или сфере) с радиусом ***рслп*** и центром в нулевой точке. Для этого используют таблицу равномерно распределенных случайных чисел. Пусть, например, принято *рслп=* 15. В таблице случайных чисел случайным образом выбирают начало отсчета, и находят первое попавшееся число из интервала (0,15), скажем 5 (оно находится в 6-м столбце и 9-й строке).

В том случае, когда число факторов п=2, вторая составляющая оценивается однозначно по теореме Пифагора:

https://studref.com/htm/img/39/8326/233.png

Знак перед устанавливают также по таблице случайных чисел: если после числа 5 в столбце стоит нечетное число (в данном случае - это 03), то ставят минус, т.е. <^3= V152 -52 = -14. Если бы после числа 5 было четное число, то перед |а стоял бы плюс.

https://studref.com/htm/img/39/8326/234.png

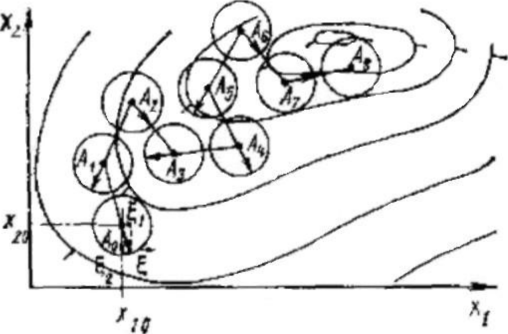
(где к= 1,2, ..., п-2) и устанавливая их знаки по предыдущему случайному числу. Последнюю составляющую определяют однозначно по теореме Пифагора:

https://studref.com/htm/img/39/8326/235.png

знак устанавливают по числу, следующему в таблице за ***%пЛ.*** Составляющую откладывают от начальной точки А0 (в соответствии с присвоенным ей знаком) в отрицательном или положительном направлении от начальной точки параллельно своей факторной оси.

В начальной точке А0 и в точке, служащей концом вектора ***%***, выполняют пробные опыты, полученные значения отклика сравнивают. Если отклик в исходной точке меньше, чем в конце вектора %, то совершают рабочий шаг по направлению этого вектора, а если, наоборот, в начальной точке А0 отклик больше, чем в конце вектора, то рабочий шаг делают в противоположном направлении. Размер рабочего шага ***аслп*** обычно выбирают из условия:

https://studref.com/htm/img/39/8326/236.png



**public static OptimalResult RandomMethod(Vector initialVector, double h, double eps, Func<Vector, double> func)**

Входные параметры:

* **initialVector**: Начальная точка, с которой начинается процесс оптимизации.
* **h**: Начальный шаг для обновления точек (может быть скорректирован в процессе).
* **eps**: Заданная точность, при достижении которой оптимизация завершается.
* **func**: Многомерная функция, которую мы оптимизируем.

Шаги метода:

* **Инициализация переменных:**
* Инициализация начальной точки **initialVector**.
* Инициализация текущей точки **currentSolution**.
* Инициализация текущего и лучшего значения функции **currentObjectiveValue** и **bestObjectiveValue**.
* Инициализация массива для хранения кандидатов **value**.
* Инициализация счетчика итераций **iterationCount**.
* **Итерационный процесс:**
* Повторение итераций до тех пор, пока шаг **h** больше заданной точности **eps**.
* На каждой итерации выполняются следующие шаги:
* Генерация нескольких случайных кандидатов **value** вокруг текущей точки **initialVector**.
* Вычисление значений функции для каждого кандидата.
* Выбор лучшего кандидата, у которого значение функции минимально.
* Обновление текущей точки **currentSolution** и текущего значения функции **currentObjectiveValue** на основе лучшего кандидата.
* Обновление шага **h** в зависимости от того, уменьшилось ли значение функции относительно предыдущего шага.
* Обновление начальной точки **initialVector** для следующей итерации.
* **Возврат результата:**
* Возврат результатов оптимизации в виде объекта **OptimalResult**, который содержит:
* **currentSolution**: конечная точка оптимизации.
* **gradient**: градиент в конечной точке (в данном коде не вычисляется, указано как **null**).
* **finalObjectiveValue**: значение функции в конечной точке.
* **iterationCount**: количество выполненных итераций.

**Метод случайного поиска с ограничениями.**

Метод случайного поиска с ограничениями осуществляет поиск оптимума функции в пространстве переменных с учетом заданных ограничений. Этот метод случайного поиска с ограничениями осуществляет случайную генерацию точек в пространстве переменных, учитывая заданные простые скалярные ограничения и функциональные ограничения для каждой переменной.  
Метод случайного поиска с ограничениями использует следующие виды ограничений:  
1. Простые скалярные ограничения (`Restriction`):  
• `DoubleOgr`: Этот тип ограничения представляет ограничение на переменную, которая должна находиться в диапазоне между заданными значениями `Upper` (верхнее ограничение) и `Lower` (нижнее ограничение).  
• `UpperOgr`: Переменная должна быть меньше или равна значению `Upper`.  
• `LowerOgr`: Переменная должна быть больше или равна значению `Lower`.  
2. Функциональные ограничения (`RestrictionFunc`):  
• Функциональные ограничения позволяют ограничивать значения функций, зависящих от вектора переменных. Они предоставляют возможность проверять ограничения для значений, полученных в результате вычисления функции по вектору переменных.  
• Также поддерживаются типы ограничений: `DoubleOgr`, `UpperOgr` и `LowerOgr`.

**public static OptimalResult FindMinRandomMethodWithRestrictions(Vector initialPoint, double h, double eps,  Func<Vector, double> func, RestrictionFunc[] restrictions, Restriction[] restrArg)**

* Определяется начальная точка и количество попыток (trials).
* Выполняется проверка, соответствует ли начальная точка всем ограничениям. Если нет, то она корректируется с помощью функции проекции.
* Затем начинается цикл итераций, в течение которого генерируются случайные точки и проверяются на соответствие ограничениям.
* Для каждой успешно сгенерированной точки вычисляется значение целевой функции.
* Найдена точка с наименьшим значением функции становится текущей точкой для следующей итерации.
* Если значение целевой функции для текущей точки больше предыдущего значения, то шаг уменьшается в два раза, иначе - увеличивается в полтора раза.
* Когда достигнута требуемая точность (разница между текущим и предыдущим значением функции меньше заданного порога), алгоритм завершается.

Метод называется **FindMinRandomMethodWithRestrictions** и принимает несколько параметров:

* **initialPoint**: Начальная точка для процесса оптимизации.
* **h**: Размер шага или масштабирующий коэффициент, используемый на каждой итерации.
* **eps**: Уровень точности или допустимой погрешности для сходимости.
* **func**: Целевая функция, которую необходимо минимизировать.
* **restrictions**: Массив экземпляров **RestrictionFunc**, представляющих ограничения на оптимизацию.
* **restrArg**: Массив экземпляров **Restriction**, предоставляющих дополнительную информацию для обработки ограничений.

**Основные шаги:**

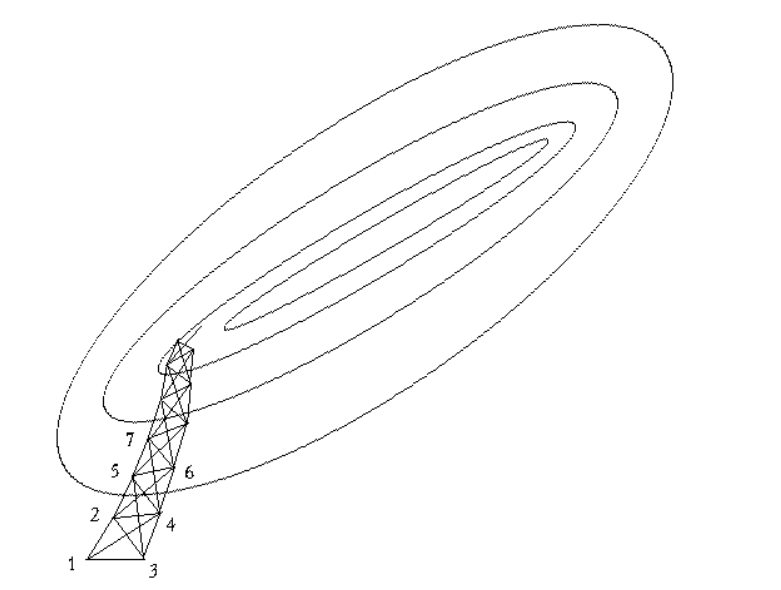
* **Инициализация:**
* Вычисляется начальное значение целевой функции для стартовой точки.
* Начальная точка корректируется так, чтобы удовлетворять ограничениям путем проецирования ее на допустимую область.
* **Основной цикл:**
* Метод выполняется до тех пор, пока размер шага **h** не станет меньше заданной точности **eps**.
* Внутри цикла генерируются несколько пробных точек путем добавления случайного вектора (уменьшенного на **h**) к текущей точке.
* Каждая пробная точка проверяется на соответствие ограничениям. Если пробная точка нарушает какое-либо ограничение, генерируется новая пробная точка.
* Для каждой допустимой пробной точки вычисляется значение целевой функции.
* Выбирается пробная точка с минимальным значением целевой функции в качестве следующей текущей точки.
* Размер шага **h** корректируется в зависимости от того, увеличилось или уменьшилось значение целевой функции.
* **Завершение:**
* Когда цикл сходится (размер шага **h** падает ниже уровня допустимой погрешности), получается окончательная оптимальная точка.
* Вычисляются значения ограничений и значения целевой функции в оптимальной точке.
* Возвращается объект **OptimalResult**, содержащий оптимальную точку, значения ограничений, значения целевой функции и количество итераций.

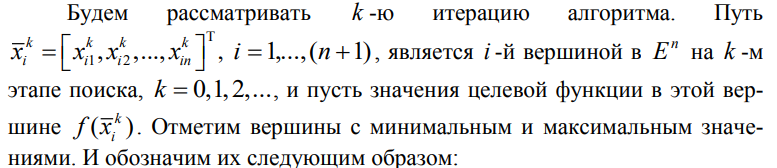
**Метод нелдера-мида**

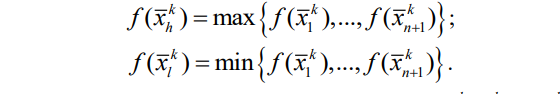
Метод Нелдера-Мида (Nelder-Mead) является итерационным численным методом оптимизации, применяемым для поиска минимума (или максимума) многомерных функций без использования их производных.

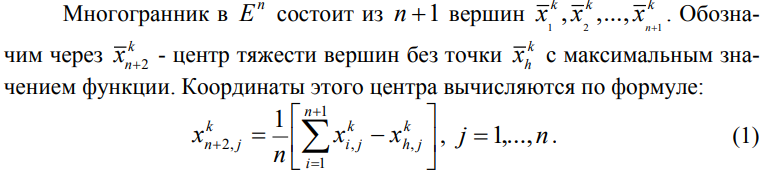
В самом простом виде симплексный алгоритм заключается в следующем. Строится регулярный симплекс. Из вершины, в которой *f(x)* максимальна (точка 1, см. рис. 2.4) проводится проектирующая прямая через центр тяжести симплекса. Затем точка 1 исключается и строится новый отраженный симплекс из оставшихся старых точек и одной новой, расположенной на проектирующей прямой на надлежащем расстоянии от центра тяжести.

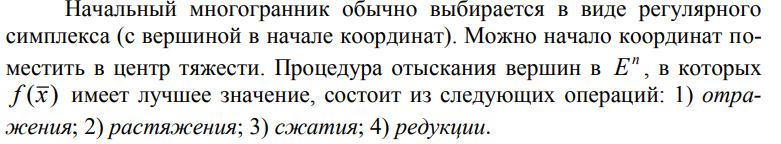
Продолжение этой процедуры, в которой каждый раз исключается вершина, где целевая функция максимальна, а также использование правил уменьшения размера симплекса и предотвращения циклического движения в окрестности экстремума, позволяет достаточно эффективно определять минимум для "хороших" функций. Но для "овражных" функций такой поиск неэффективен. Представление об идее алгоритма дает рисунок 2.4. В симплексном алгоритме Нелдера и Мида минимизация функций n переменных осуществляется с использованием деформируемого многогранника.

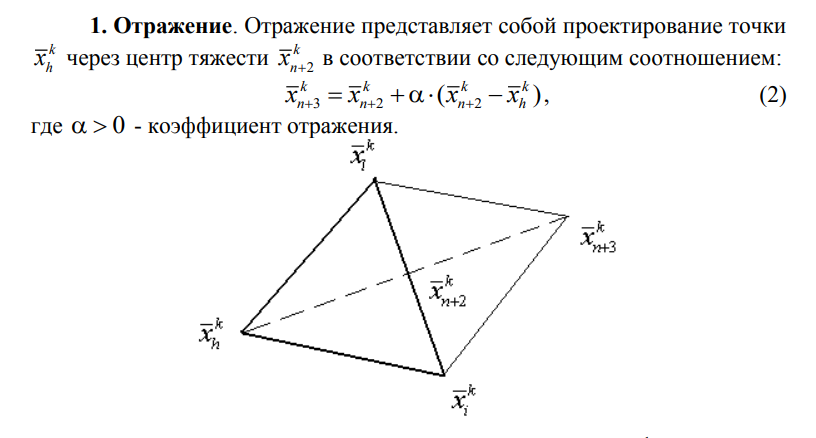


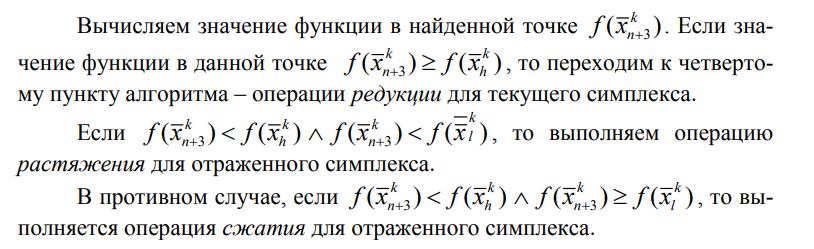


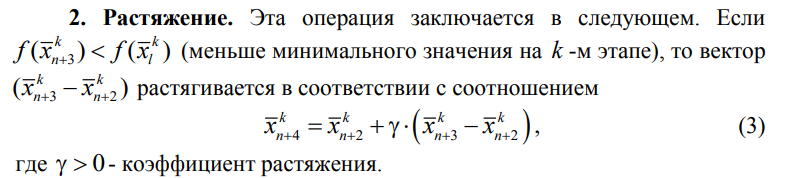


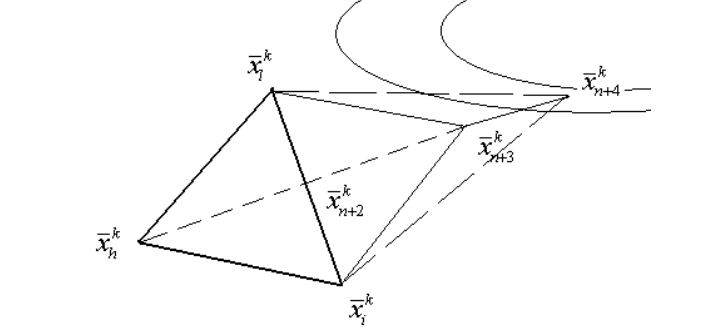


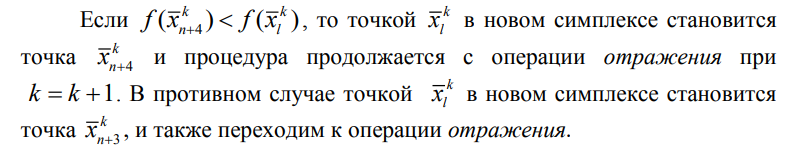


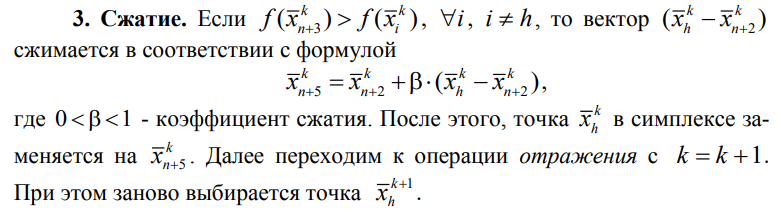


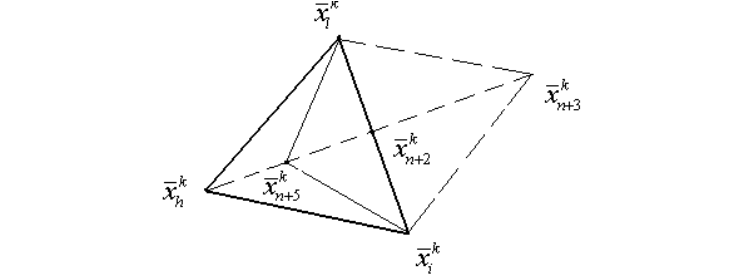


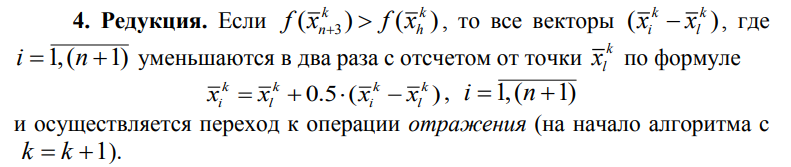


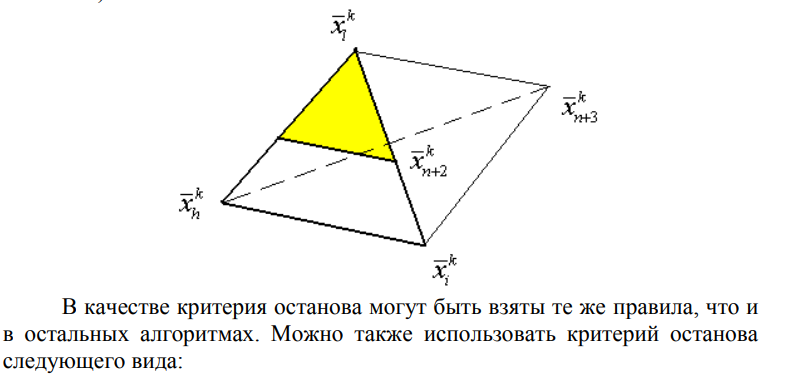


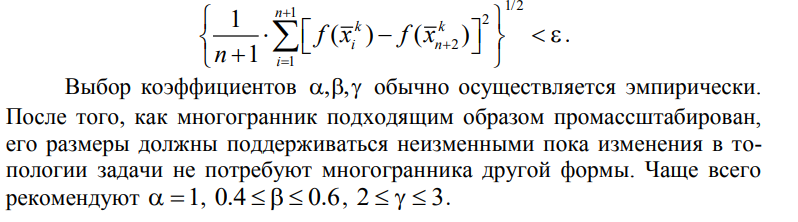












**Входные параметры:**

1. **xn** (тип: **Vector**): Начальная точка оптимизации, представленная вектором.
2. **h** (тип: **double**): Параметр шага, используемый для создания симплекса вокруг начальной точки.
3. **eps** (тип: **double**): Параметр точности, представляющий требуемую точность метода оптимизации.
4. **f** (тип: **Func<Vector, double>**): Функция, которую требуется оптимизировать. Принимает вектор в качестве входного параметра и возвращает значение функции в этой точке.

**Выходные параметры:**

1. **result** (тип: **Vector**): Вектор, представляющий найденную оптимальную точку согласно методу Нелдера-Мида. Эта точка соответствует локальному минимуму функции **f**.

**public static Vector NelderMeadeMethod(Vector xn, double h, double eps, Func<Vector, double> f)**

Ключевые компоненты метода:

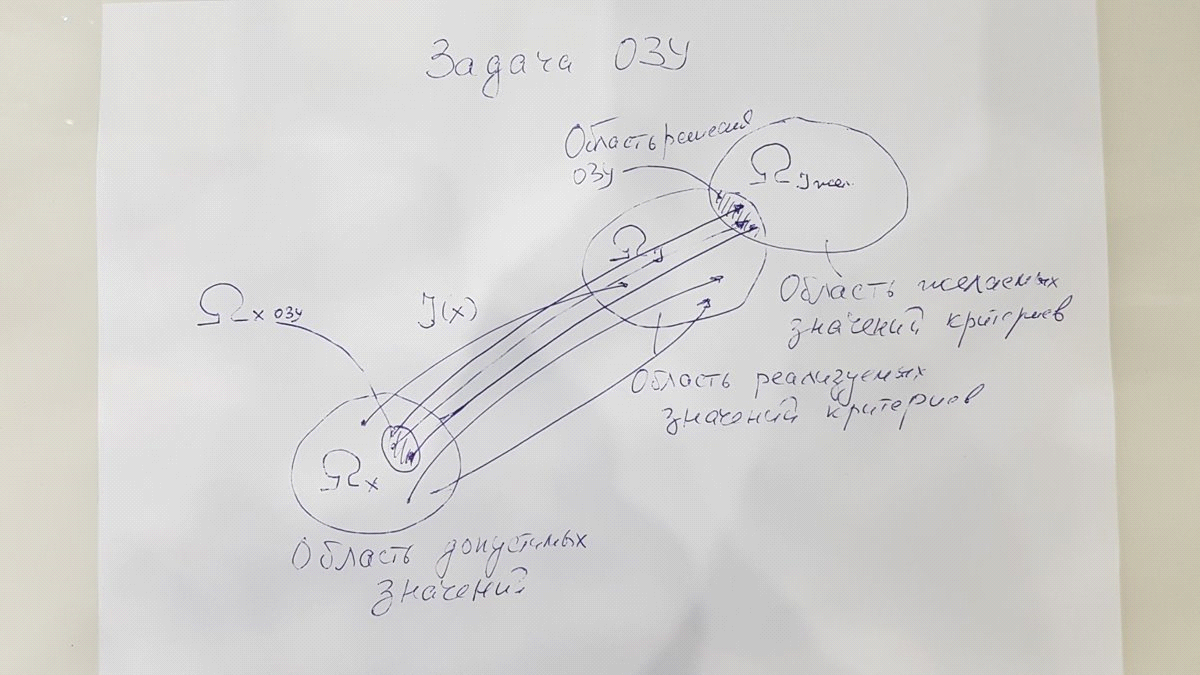
* **Инициализация:**
* Начальная точка **xn** инициализирует первую вершину симплекса.
* Параметры **alpha**, **beta**, и **gamma** используются для управления операциями отражения, сжатия и растяжения соответственно.
* **Формирование симплекса:**
* Формируется симплекс, состоящий из **n + 1** точек вокруг начальной точки.
* **Итеративный процесс:**
* На каждой итерации оцениваются значения целевой функции во всех точках симплекса.
* Точки сортируются по значениям целевой функции, и выбираются лучшая, вторая по худшей и худшая точки.
* Вычисляется центр масс между лучшей и второй по худшей точками (midpoint).
* **Операции отражения, растяжения и сжатия:**
* Операция отражения: создание отраженной точки относительно центра масс.
* Операция растяжения: создание точки, далеко вытянутой вдоль линии отражения.
* Операция сжатия: создание точки, сжатой в сторону центра масс между лучшей и худшей точками.
* **Обновление симплекса:**
* Обновление точек симплекса в соответствии с результатами операций.
* Повторение процесса до достижения критерия сходимости (размера симплекса меньше заданной погрешности).

Особенности и Наблюдения:

* **Отражение и Эксплорация:** Операция отражения способствует исследованию пространства параметров. В случае успешного отражения, симплекс расширяется в направлении улучшения.
* **Сжатие и Эксплойтация:** Операция сжатия осуществляет эксплуатацию найденных улучшений, уменьшая размер симплекса вокруг наилучшей точки.
* **Параметры метода:** Параметры **alpha**, **beta** и **gamma** могут потребовать настройки в зависимости от конкретной задачи.

**ОЗУ**

Основная идея метода ОЗУ заключается в том, что точку каждого пробного опыта для изучения поверхности отклика в районе базовой (начальной) точки выбирают случайным образом (отсюда и название метода). Но в отличие от Метода случайного поиска, мы вычисляем не просто значение функции, а значение нормирования функции, а зависимости нормирования функции, в зависимости от типа ограничений.



Основная задача управления(ОЗУ) предполагает поиск области аргументов с учетом их ограничений, при которых критерий принимает желаемые значения.

**Алгоритм метода:**

1. Задаются желаемые значения критерия в виде ограничений. Ограничения могут быть как односторонние, так двухсторонние
2. Подбираются аргументы , при которых функция будет принимать желаемые значения. Для этой задачи можно выбрать метод случайного поиска, методом деформированного многогранника и т.д.
3. Частные критерии приводятся к безразмерной форме и нормированию:

* Если ограничение двухстороннее, то нормирующих функций будет две: и
* Если ограничение снизу, и оно больше нуля, то нормирующая функция
* Если ограничение снизу, и оно меньше нуля, то нормирующая функция
* Если ограничение снизу, и оно равняется нулю, то нормирующая функция . Здесь задается при постановке задачи
* Если ограничение сверху, и оно больше нуля, то нормирующая функция
* Если ограничение сверху, и оно меньше нуля, то нормирующая функция
* Если ограничение сверху, и оно равняется нулю, то нормирующая функция . Здесь задается при постановке задачи

Во всех приведенных выше случаях, если значение нормирующей функции ( и , то значения частного критерия лежит в допустимой области.

Метод **Нормирования** представляет собой функцию для нахождения нормы вектора ограничений и критерия g(x) на основе заданных ограничений. Этот метод используется в контексте задач управления или оптимизации, где вектор x представляет собой набор переменных, а RestrictionFunc[] ogr - массив ограничений.

1. Вводится обобщенный нормированный критерий g(x), который является максимальным значением из всех значений нормирующих функций. Если значение g(x) , то тогда при любом j , и значения всех частных критерий находятся в допустимой области. То есть в таком случае задачу ОЗУ можно считать решенной. В противном случае повторить шаги 2-4. При этом, если g(x) < 1, то задача имеет множество решений и желательно найти область аргументов, где g(x) . Для этого поиск можно осуществлять в окрестности точки g(x) < 1.

**Входные параметры:**

* Vector x: Вектор переменных, для которого вычисляется норма ограничений.
* RestrictionFunc[] constraintFunctions: Массив объектов ограничений, содержащих информацию о типе ограничения, его верхней и нижней границах, а также функцию для вычисления значения ограничения.

**Выходные параметры:**

* double: Максимальная норма среди всех ограничений.

**Основные идеи и принцип метода:**

1. **Проекция на область допустимых значений:** Начальная точка и все новые точки, генерируемые в процессе оптимизации, проецируются на область допустимых значений переменных. Это гарантирует, что решение останется в пределах допустимой области.
2. **Условия удовлетворения ограничений:** Проверка удовлетворения ограничений, представленных в виде функций *ph*. Если новая точка не удовлетворяет ограничениям, она корректируется до тех пор, пока не будет удовлетворять всем ограничениям.
3. **Использование случайных точек:** В каждой итерации генерируются случайные точки вокруг текущей точки. Это позволяет методу исследовать различные направления для поиска оптимума.

**Итеративное обновление:** Процесс обновления текущей точки и шага повторяется до достижения заданной точности (*eps*). Шаг (ℎ*h*) увеличивается, если удалось найти точку с меньшим значением

**Входные параметры:**

* initialVector- начальное приближение (вектор переменных),
* *h* - начальный шаг оптимизации,
* *eps* - заданная точность оптимизации,
* restrArg- массив ограничений на переменные,
* restrValue- массив ограничений в виде функций,
* func - массив целевых функций.

**Выходные параметры:**

* Возвращается объект типа OptimumResult, содержащий оптимальное значение переменных (*xt*), значения целевых функций (*fopt*), информацию о нарушенных ограничениях (в данном случае, возвращается null), и количество итераций (*k*).

Итак, основные идеи основной задачи управления заключаются в следующем:  
• Необходимо обеспечить, чтобы система оставалась в области допустимых значений.  
• Необходимо обеспечить, чтобы система достигала желаемых значений критериев.  
• Для этого необходимо выбрать такой способ управления, который позволит системе выполнять эти два требования

**public static OptimalResult Optimize\_OZU(Vector initialVector, Restriction[] restrArg, RestrictionFunc[] restrValue, RestrictionFunc[] func, double h, double eps)**

Ключевые компоненты метода:

* **Инициализация:**
* Задается начальная точка **initialVector**.
* Выполняется проекция начальной точки на допустимую область для каждой переменной в случае её нарушения.
* **Проверка допустимости:**
* Проверяется допустимость начальной точки по всем ограничениям. Если начальная точка нарушает хотя бы одно равенство, метод завершается.
* **Основной итерационный процесс:**
* На каждой итерации выполняется цикл по всем возможным возмущениям текущей точки.
* Каждое возмущение подвергается проверке на допустимость, и в случае нарушения производится его коррекция.
* Для каждого возмущения вычисляется значение целевой функции, и выбирается лучшее из них.
* **Обновление текущей точки:**
* Если лучшее значение целевой функции улучшает текущее значение, то точка обновляется, и размер шага увеличивается.
* В противном случае размер шага уменьшается.
* **Критерий остановки:**
* Метод завершается, когда размер шага становится меньше заданной погрешности **eps**.
* **Вычисление и возврат результата:**
* Вычисляются значения целевых функций и ограничений для найденной оптимальной точки.
* Создается объект **OptimalResult**, содержащий оптимальную точку, значения ограничений и целевых функций, а также количество итераций.

**Список литературы**

* **"Методы оптимизации"** А. В. Архангельский, Ю. М. Ермолаев, А. В. Фигурин
* **"Оптимизация. Теория и практика"** Ю. Е. Нестеров
* **"Выпуклая оптимизация"** А. Б. Тихонов, А. А. Лобанов
* **"Численные методы оптимизации"** А. В. Фигурин
* **"Методы оптимизации и их применение"** В. Ф. Демьянов, А. Б. Тихонов