Министерство образования и науки Российской Федерации

Государственное образовательное учреждение высшего профессионального образования Санкт-Петербургский государственный технологический институт
(технический университет)

Кафедра систем автоматизированного проектирования и управления

И. М. Зобнин, А. С. Афанасьев, И. А. Смирнов

Методы оптимизации в среде Matlab

Санкт-Петербург
2022

УДК 004.422.8:66.011.001

Зобнин, И. М., Методы оптимизации в среде Matlab / И. М. Зобнин, А. С. Афанасьев, И. А. Смирнов – СПб.: СПбГТИ(ТУ), 2022. –

На примере целевых функций, описывающих некоторые химико-технологические системы и процессы, рассмотрена работа безусловных методов оптимизации универсального математического пакета Matlab с соответствующей двумерной и трёхмерной визуализацией, реализованной средствами этого же пакета.

Методические указания предназначены для студентов (бакалавров и магистров), изучающих курс «Методы оптимизации».

Оглавление

ВВЕДЕНИЕ

Matlab (MATrix LABoratore) – это апробированная и надёжная система, рассчитанная на решение широкого круга математических задач с представлением данных в универсальной матричной форме, предложенной фирмой MathWorks, Inc. Matlab является одновременно операционной средой и высокоуровневым интерпретируемым языком программирования. Кроме того, Matlab имеет мощную библиотеку визуализации, позволяющую строить двумерные и трёхмерные графики, что поможет в демонстрации работы алгоритмов.

Пакеты расширения Toolbox (Toolbox Optimization и Toolbox Global Optimization) специализируются на решении задач в различных предметных областях: в физике, в специальных разделах математики и математическом моделировании химико-технологических объектов.

Toolbox Optimization реализовывает алгоритмы одномерной и многомерной, безусловной и условной минимизации, линейного, булевого и квадратичного программирования, наименьших квадратов и решения уравнений, а также алгоритмы многокритериальной оптимизации.

В пакете Global Optimization используются методы прямого поиска, мультистарта, моделирования отжига, генетический алгоритм, ориентированные на поиск глобального минимума или многих минимумов.

В пособии рассматривается оптимизация методами безусловной оптимизации целевых функций, описывающих различные химико-технологические системы и процессы с соответствующей демонстрацией в виде двумерных и трёхмерных графиков.

ОДНОМЕРНАЯ МИНИМИЗАЦИЯ В MATLAB

Для одномерной оптимизации непрерывных функций предназначена функция пакета «fminbnd». Решаемая задачи имеет вид.

$$minf(x), a < x < b$$

Алгоритм минимизации базируется на методах золотого сечения и квадратичной аппроксимации (параболической интерполяции).

Варианты обращения к функции «fminbnd»:

```
x = fminbnd(fun,x1,x2)
x = fminbnd(fun,x1,x2,options)
x = fminbnd(problem)
[x,fval] = fminbnd(__)
[x,fval,exitflag] = fminbnd(__)
[x,fval,exitflag,output] = fminbnd(__)
```

Слева от знака равенства записываются выходные величины, и если их больше одной, то они перечисляются через запятую в квадратных скобках. Справа от имени функции в круглых скобках указываются входные величины (аргументы функции). В приведенных записях приняты следующие обозначения:

`x` – значение *x*, соответствующее локальному минимуму функции *f*(*x*); `fval` – значение *f*(*x*) в точке найденного локального минимума; `exitflag` – признак, идентифицирующий причину завершения алгоритма, его возможные значения:

`1` – сходимость алгоритма к минимуму;

`0` – число итераций превысило максимально установленное;

`-1` – причина в функции вывода (решение не достигнуто);

`-2` – границы несовместны.

`Output` – структура, содержащая информацию о процессе, имеет поля:

`Iterations` – число итераций;

`funcCount` – число вычислений целевой функции;

`algorithm` – используемый алгоритм;

`message` – конечное сообщение.

Входные аргументы:

`fun` – целевая функция, может быть определена как *m*-файл (*m*-фун-

кция), например:

```
function f = myfun(x)
f = sin(x^2);
```

(файл сохраняется под именем `myfun.m`), и тогда на месте `fun` записывается `@myfun`, где `myfun` – имя функции и соответствующего ей *m*-файла; также `fun` можно представить непосредственно формулой (без использования имени) как конструкцию `@(x)sin(x^2)` или как строковое выражение `'sin(x^2)'`;

`x1,x2` – значения левой и правой границ переменной *x*;

`options` – опции алгоритма, исходно все установлены по умолчанию;применяется, если необходимо изменить какие-либо параметры оптимизации; при этом перед обращением к функции `fminbnd` вносятся изменения в настройки опций `options` с помощью функции `optim- set`, аргументами которой являются пары «имя параметра, устанавливаемое значение параметра», например установка

options=optimset(‘Display’, ‘off’)

исключит вывод на экран (в командное окно) промежуточных данных. При значении `'iter'` будут выводиться результаты каждой итерации.Другие значения параметра – `'final'` и `'notify'` (вывод итогов и вывод только при отсутствии сходимости). Указание параметра `'Outputfcn'`, в качестве значения которого записывается имя функции, приводит к вызову этой функции на каждой итерации алгоритма. Значение параметра `'PlotFcn'`

определяет функцию вывода графики. В пакете это функции `@optimplotx` (рисует текущую точку), `@optimplotfunccount` (выводит количество вычислений функции), `@optimplotfval` (выводит текущее значение целевой функции). Функции вывода и графики могут быть написаны пользователем. При включении параметра `FunValCheck ('on')` выводится сообщение привозврате целевой функцией комплексного значения или `Inf` и `NaN`.

`Problem` – сохраненное из `GUI Optimization Tool` описание задачи, а именно:

`f` – целевая функция,

`x1` – левая граница,

`x2` – правая граница,

`solver` – `'fminbnd'`,

`options` – структура параметров, создается `optimset`.

Пример 1.

Объектом оптимизации является химико-технологическая система, состоящая из двух реакторов непрерывного действия. В них в результате химического взаимодействия из двух сырьевых компонентов, объемные расходы которых A1 и A2 (м^3/ч), образуется целевой компонент в количестве C (кг/ ч). Для исследования процесса разработана эмпирическая математическая модель, в соответствии с которой количество C зависит от объемных расходов компонентов по следующему правилу:

$$C = \alpha (A1^2 + \beta A2 - \mu V1)^N + \alpha 1(\beta 1 A1 + A2^2 - \mu 1 V2)^N,$$

где α, α1, β, β1, μ, μ1 – нормирующие коэффициенты, равные 1;

N – количество реакторов (2 шт.);

V1 и V2 – рабочие объемы реакторов (11 и 7 м^3 соответственно).

Технологическим регламентом установлены следующие требования к проведению процесса. Объемные расходы сырьевых компонентов A1 и A2 могут изменяться в диапазоне от 1 до 10 м^3/ч соответственно; кроме того, необходимо, чтобы суммарная производительность реакторов была не больше 8 м^3 /час.

Необходимо найти такие условия проведения процесса (значения A1 и A2), при которых обеспечивается максимальный выход целевого компонента в кг в сутки (за 24 часа). Точность решения – 0,1 м^3/ч.

Из-за того, что функция «fminbnd» поддерживает лишь одномерную оптимизацию, придётся сделать варьируемый параметр A1 входным и задать ему константное значение, например 1. Кроме того, ограничение суммарной производительности тоже учитываться не будет. Также, чтобы формально выразить целевую функцию и формализовать задачу, необходимо от обозначения A2 перейти к стандартному X2.

Тогда формализованная задача оптимизации может быть поставлена следующим образом:

$$\begin{cases} C(X2) = \alpha * (A1^2 + \beta * X2 - \mu * V1)^N + \alpha 1 * (\beta 1 * A1 + X2^2 - \mu 1 * V2)^N \rightarrow max \\ \alpha = \alpha 1 = \beta = \beta 1 = \mu = \mu 1 = 1 \\ N = 2 \\ V1 = 11 \\ V2 = 7 \\ A1 = 1 \\ 1 \leq A2 \leq 10 \end{cases}$$

Шаг 1. Создадим файл скрипта. На вкладке «Home» есть соответствующая кнопка (Рисунок 1).

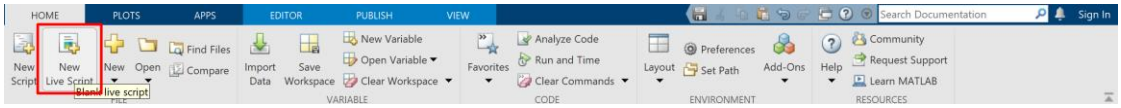

Рисунок 1 – Создание скрипта MATLAB

Поместим в него целевую функцию. Для этого напишите следующий код:

```
function C = TargetFunc(a,A1,B,A2,u,V1,N,a1,B1,u1,V2)
C=a*(A1^2+B*A2-u*V1).^N+a1*(B1*A1+A2.^2-u1*V2).^N;
end
```

Примечание. Возведению в степень N первой и второй скобок предшествует точка, что означает поэлементное действие, иначе MATLAB будет выполнять матричную операцию и выдаст ошибку.

Шаг 2. Создадим функцию построения графика функции «FuncPlot». Для этого напишите следующий код:

```
function FuncPlot(a,A1,B,X2l,X2u,u,V1,N,a1,B1,u1,V2)
x=X2l:0.1:X2u;
y=TargetFunc(a,A1,B,x,u,V1,N,a1,B1,u1,V2);
plot(x,y);
xlabel('x');
ylabel('y');
title('Minimization by fminbnd');
hold on;
end
```

В 1-й строке мы задаём массив значений варьируемого параметра от X2l до X2u с шагом 0,1. На его основе создаём массив значений целевой функции во 2-й строке, вызывая целевую функцию. В 3-й строке функцией plot(x,y) создаём график функции на основе двух предыдущих массивов. С помощью функций «xlabel», «ylabel», «title» на 4, 5, 6 строках задаём подписи к осям и наименование графика соответственно.

Шаг 3. Создадим функцию траектории движения метода оптимизации «AddSolutionToPlot».

```
function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
end
```

x, y – массивы значений варьируемого параметра и целевой функции. В первой строке мы «удерживаем» созданный во 2-м шаге график, иначе функция «plot» на следующей строке создаст новый. Функция «text» на 3-й строке отмечает на графике начальную точку.

Шаг 4. Создадим функцию для начала процесса оптимизации «Optim». Она будет содержать следующий код:

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,A1,B,X2l,X2u,u,V1,N,a1,B1,u1,V2)
    fValuesSolution=[];
    xValuesSolution=[];
    options=optimset('outputfcn', @outfun, 'TolX', 0.1);
    [x,fval,exitflag,output] = fminbnd(@(X2)-
TargetFunc(a,A1,B,X2,u,V1,N,a1,B1,u1,V2), X2l, X2u, options);
    F=-fval;
end
```

x – оптимальное значение A2, F – максимальное значение целевой функции, xValuesSolution – значения варьируемого параметра, в которых «был» метод оптимизации, то же и fValuesSolution, только это значения целевой функции. options – настройка метода оптимизации, в которой задана функция, к которой метод будет обращаться в каждую итерацию (`'outputfcn', @outfun`), а также точность по варьируемому параметру (`'TolX', 0.1`). Так как методы оптимизации в MATLAB занимаются минимизацией целевой функции, необходимо передать нашу целевую функцию со знаком «-» `fminbnd(@(X2)-TargetFunc(a,A1,B,X2,u,V1,N,a1,B1,u1,V2), X2l, X2u, options)` – вызов метода оптимизации, где `@(X2)-TargetFunc(a,A1,B,X2,u,V1,N,a1,B1,u1,V2)` – вызов отрицательной целевой функции со значением варьируемого параметра (X2), а также значениями входных параметров; `X2l, X2u, options` – нижняя и верхняя границы варьирования, а также настройка метода оптимизации соответственно. `F=-fval` – присвоение оптимального значения целевой функции.

Шаг 5. Создадим функцию фиксирования значений варьируемого параметра и целевой функции «outfun» после `F=-fval` внутри функции «Optim». Она будет содержать следующий код:

```
    function stop = outfun(x, optimValues, state)
```

```
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -
optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
```

x – значение варьируемого параметра; optimValues.fval – значение целевой функции; stop-false – флаг, который указывает на остановку работа метода оптимизации, в нашем случае всегда равный false, так как мы не хотим вручную останавливать работу метода.

Шаг 6. В начале скрипта зададим значения входных параметров, границы варьирования A2. Для этого вставим следующий код:

```
a=1;A1=1;B=1;u=1;V1=11;N=2;a1=1;B1=1;u1=1;V2=7;
A2l=1;A2u=10;
FuncPlot(a,A1,B,A2l,A2u,u,V1,N,a1,B1,u1,V2);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,A1,B,A2l,A2u,u,V1,N,a1,B1,u1,V2);
AddSolutionToPlot(xValuesSolution, fValuesSolution);
exitflag
output
x
F
```

В первой строке мы присваиваем значения входных параметров. Во второй – границы варьирования A2. FuncPlot – строим график целевой функции. Далее вызываем нашу функцию для начала процесса оптимизации. Получив вывод этой функции, нанесём траекторию движения метода оптимизации на график целевой функции вызовом «AddSolutionToPlot», выведем данные метода оптимизации (флаг выхода, другие данные, оптимальное значение A2, максимальное значение целевой функции).

Листинг кода скрипта представлен ниже:

```
a=1;A1=1;B=1;u=1;V1=11;N=2;a1=1;B1=1;u1=1;V2=7;
A2l=1;A2u=10;
FuncPlot(a,A1,B,A2l,A2u,u,V1,N,a1,B1,u1,V2);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,A1,B,A2l,A2u,u,V1,N,a1,B1,u1,V2);
```

```
AddSolutionToPlot(xValuesSolution, fValuesSolution);
exitflag
output
x
F
```

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,A1,B,X2l,X2u,u,V1,N,a1,B1,u1,V2)
    fValuesSolution=[];
    xValuesSolution=[];
    options=optimset('outputfcn', @outfun, 'TolX', 0.1);
    [x,fval,exitflag,output] = fminbnd(@(X2)-
TargetFunc(a,A1,B,X2,u,V1,N,a1,B1,u1,V2), X2l, X2u, options);
    F=-fval;
    function stop = outfun(x, optimValues, state)
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -
optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
end

function C = TargetFunc(a,A1,B,A2,u,V1,N,a1,B1,u1,V2)
C=a*(A1^2+B*A2-u*V1).^N+a1*(B1*A1+A2.^2-u1*V2).^N;
end


function FuncPlot(a,A1,B,X2l,X2u,u,V1,N,a1,B1,u1,V2)
x=X2l:0.1:X2u;
y=TargetFunc(a,A1,B,x,u,V1,N,a1,B1,u1,V2);
plot(x,y);
xlabel('x');
ylabel('y');
title('Minimization by fminbnd');
hold on;
end


function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
```

```
end
```

Примечание. После переносе кода в скрипт MATLAB, выделите его сочетанием клавиш Ctrl+A, нажмите правую клавишу мыши и в выпадающем меню выберите «Convert Between Code and Text».

При нажатии кнопки Run (Рисунок 2) выводится график (Рисунок 3), а также вывод метода оптимизации.

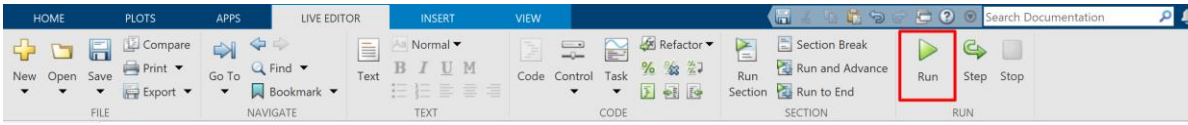

Рисунок 2 – Расположение кнопки запуска скрипта

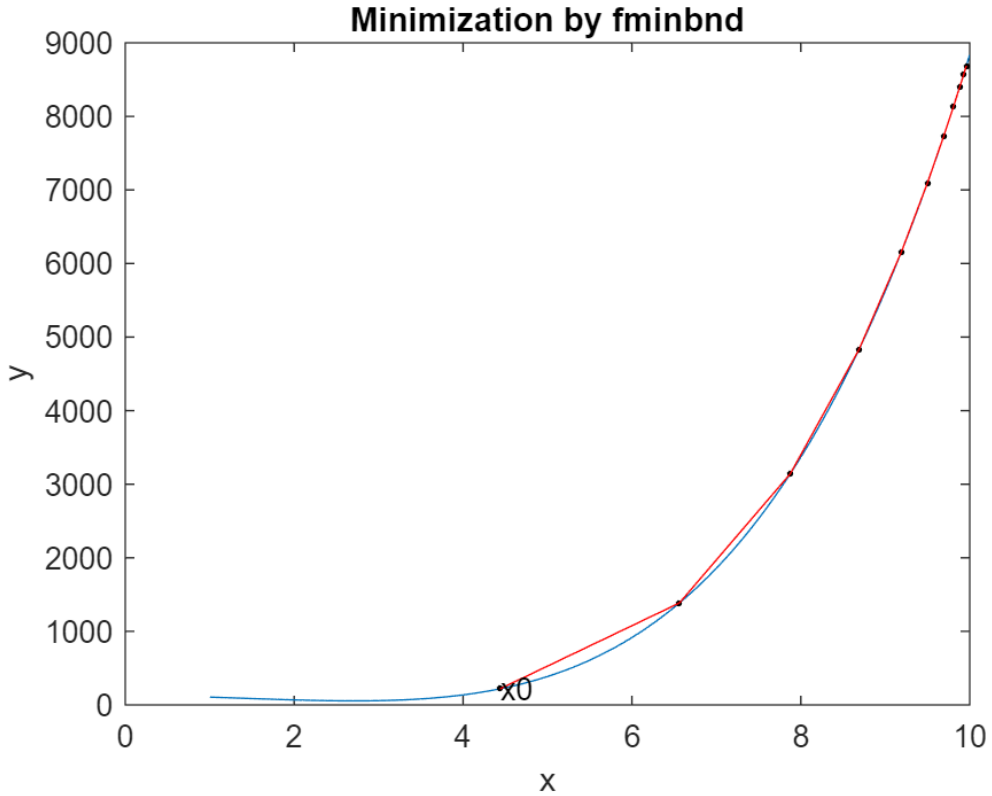

Рисунок 3 – Траектория движения метода оптимизации

```
exitflag = 1
output = struct with fields:
    iterations: 10
     funcCount: 11
     algorithm: 'golden section search, parabolic interpolation'
       message: 'Optimization terminated:↵ the current x
satisfies the termination criteria using OPTIONS.TolX of
1.000000e-01 ↵'
x = 9.9602
F = 8.6871e+03
```

Пример 2.

Объектом оптимизации является процесс фильтрования с использованием установки, имеющей две фильтрационные перегородки, на каждой из которых поддерживается свой температурный режим. Известно, что объем фильтрата V(м^3/ч) связан с температурами Т1 и Т2 на каждой перегородке следующим образом:

$V = \alpha \cdot (T1 - \beta \cdot \Delta p1) \cdot \cos(\gamma \cdot \Delta p2\ sqrt(T1^{\wedge} N + T2^{\wedge}N)$,

где $\Delta p1$ и $\Delta p2$ – величина перепада давлений на каждой перегородке (Кпа);

$\Delta p1 = \Delta p2 = 1$;

α, β, γ – нормирующие множители; $\alpha = \beta = 1$; $\gamma = 3.14$;

N – количество перегородок (2 шт.).

Для эффективного фильтрования необходимо, чтобы температура на первой перегородке была не ниже -3 °С и не выше 0 °С, а на второй – не выше 3 °С и не ниже -0,5 °С, кроме того, должно выполняться условие: Т2 – Т1 ≥ 3°С.

Необходимо определить такой температурный режим проведения процесса (значения Т1 и Т2), при котором обеспечивается максимальный выход фильтрата в м^3 в сутки (24 часа). Точность решения – 0,01 °С.

Снова делаем один из варьируемых параметров входным. Присвоим Т1 значение в -3°С. Ограничение 2-о рода также учитываться не будет. Также, чтобы формально выразить целевую функцию и формализовать задачу, необходимо от обозначения Т2 перейти к стандартному Х2.

Тогда формализованная задача оптимизации может быть поставлена следующим образом:

$$\begin{cases} V(X2) = a * (T1 - \beta * \Delta p1) * \cos(\gamma * \Delta p2 * \sqrt{T1^N + X2^N}) \rightarrow max \\ \Delta p1 = \Delta p2 = \alpha = \alpha 1 = \beta = 1 \\ \gamma = 3{,}14 \\ N = 2 \\ T1 = -3 \\ -0{,}5 \leq T2 \leq 3 \end{cases}$$

Шаг 1. Изменим целевую функцию. Функция «TargetFunc» в таком случае будет содержать следующий код:

```
function V = TargetFunc(a,T1,B,p1,y,p2,T2,N)
V=a*(T1-B*p1)*cos(y*p2*sqrt(T1^N+T2.^N));
end
```

Шаг 2. Изменим аргументы функции построения графика целевой функции «FuncPlot». Это изменит первые три строки функции:

```
function FuncPlot(a,T1,B,p1,y,p2,X2l,X2u,N)
x=X2l:0.1:X2u;
y=TargetFunc(a,T1,B,p1,y,p2,x,N);
```

Шаг 3. Изменим аргументы функции процесса оптимизации «Optim», а также точность по аргументу с 0,1 на 0,01. Первые пять строк функции:

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,T1,B,p1,y,p2,X2l,X2u,N)
    fValuesSolution=[];
    xValuesSolution=[];
    options=optimset('outputfcn', @outfun, 'TolX', 0.01);
    [x,fval,exitflag,output] = fminbnd(@(T2)-
TargetFunc(a,T1,B,p1,y,p2,T2,N), X2l, X2u, options);
```

Шаг 4. Изменим значения входных параметров, границы варьируемых, а также вызов функции «Optim»:

```
p1=1;p2=1;a=1;a1=1;B=1;y=3.14;N=2;T1=-3;
T2l=-0.5;T2u=3;
FuncPlot(a,T1,B,p1,y,p2,T2l,T2u,N);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,T1,B,p1,y,p2,T2l,T2u,N);
AddSolutionToPlot(xValuesSolution, fValuesSolution);
exitflag
output
x
```

```
F
```

Далее представлен листинг кода скрипта.

```
p1=1;p2=1;a=1;a1=1;B=1;y=3.14;N=2;T1=-3;
T2l=-0.5;T2u=3;
FuncPlot(a,T1,B,p1,y,p2,T2l,T2u,N);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,T1,B,p1,y,p2,T2l,T2u,N);
AddSolutionToPlot(xValuesSolution, fValuesSolution);
exitflag
output
x
F
```

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,T1,B,p1,y,p2,X2l,X2u,N)
    fValuesSolution=[];
    xValuesSolution=[];
    options=optimset('outputfcn', @outfun, 'TolX', 0.01);
    [x,fval,exitflag,output] = fminbnd(@(T2)-
TargetFunc(a,T1,B,p1,y,p2,T2,N), X2l, X2u, options);
    F=-fval;
    function stop = outfun(x, optimValues, state)
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -
optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
end


function V = TargetFunc(a,T1,B,p1,y,p2,T2,N)
V=a*(T1-B*p1)*cos(y*p2*sqrt(T1^N+T2.^N));
end


function FuncPlot(a,T1,B,p1,y,p2,X2l,X2u,N)
x=X2l:0.1:X2u;
y=TargetFunc(a,T1,B,p1,y,p2,x,N);
plot(x,y);
xlabel('x');
ylabel('y');
```

```
title('Minimization by fminbnd');
hold on;
end


function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
end
```

Далее представлен вывод метода оптимизации.

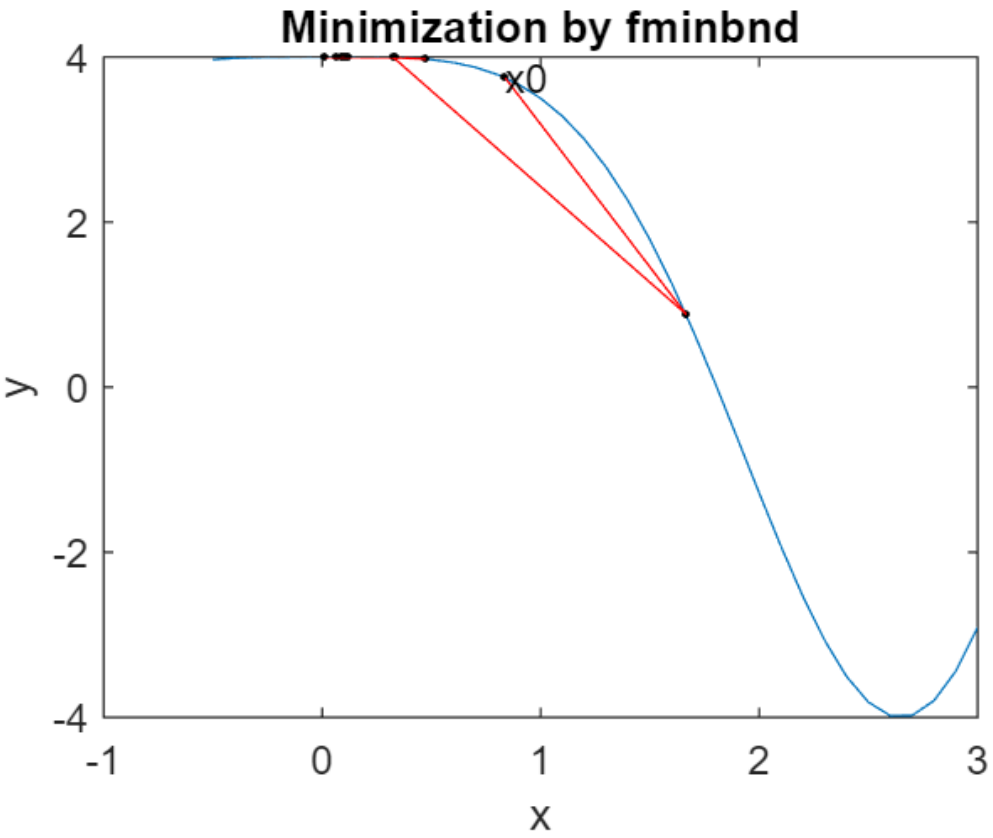

Рисунок 4 – Траектория движения метода оптимизации

```
exitflag = 1
output = struct with fields:
    iterations: 13
     funcCount: 14
     algorithm: 'golden section search, parabolic interpolation'
```

```
       message: 'Optimization terminated:↵ the current x
satisfies the termination criteria using OPTIONS.TolX of
1.000000e-02 ↵'
x = 0.0952
F = 4.0000
```

БЕЗУСЛОВНАЯ ОПТИМИЗАЦИЯ ФУНКЦИЙ МНОГИХ ПЕРЕМЕННЫХ

Для поиска минимума нелинейной функции многих переменных при отсутствии ограничений можно использовать функции `fminunc` или `fminsearch` из пакета `Toolbox Optimization`. Рассмотрим каждую из них в отдельности.

Предварительно заметим, что в MATLAB принято разделять алгоритмы оптимизации на алгоритмы для задач большой (`Large Scale`) и средней размерности (`Medium Scale`), причем это сделано довольно условно. Основное отличие в том, что `Large Scale` учитывает разреженность матрицы и для ее хранения и операций с ней использует алгебру разреженных матриц. Алгоритмы `Medium Scale` имеют дело с полными матрицами и оперируют соответствующей алгеброй. При большой размерности задачи это требует много памяти и времени. Рекомендуется сначала выбирать `Medium Scale`, так как эти алгоритмы имеют более широкие функциональные возможности.

Функция `fminunc`

По умолчанию функция `fminunc` настроена на алгоритм `Large Scale`. Он базируется на методе доверительных областей (`Trust-region method`), т.е. последовательной аппроксимации целевой функции в окрестности текущей точки более простой функцией, по которой находят следующее приближение к минимуму исходной функции. Алгоритм `Large Scale` требует задания градиента целевой функции, иначе `fminunc` будет использовать `Medium Scale`. Чтобы явно указать `Medium Scale`, следует в качестве аргумента функции `optimset` записать `'largescale','off'`.

В алгоритме Medium Scale можно выбрать один из трех методов: квази-ньютоновкий метод BFGS (Бройдена – Флетчера – Голдфарба – Шэнно) или DFP (Дэвидона – Флетчера – Пауэлла) и метод наискорейшего спуска (steepest descent method). По умолчанию используется метод BFGS, а для указания другого метода в опции нужно добавить параметр 'HessUpdate' со значением 'dfp' или 'steepdesc'.

Большинство параметров являются общими для Large Scale и Medium Scale, но часть присуща только одному из них. Варианты обращения к функции fminunc:

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(problem)
[x,fval] = fminunc(__)
[x,fval,exitflag,output] = fminunc(__)
[x,fval,exitflag,output,grad,hessian] = fminunc(__)
```

Смысл обозначений выходных аргументов:

x и fval – вектор значений искомых переменных и значение целевой функции;

exitflag – указывает причину завершения алгоритма, может принимать следующие значения:

1 – величина градиента меньше заданной точности TolFun, 2 – изменения *X* меньше заданной точности TolX,

3 – изменения *f* меньше заданной точности TolFun,

5 – предсказанное уменьшение *f* меньше, чем TolFun,

0 – превышено максимальное число итераций или число вычислений *f*,

−1 – алгоритм завершен функцией вывода (решение не получено);

Grad – градиент целевой функции;

Hessian – гессиан целевой функции;

Output – структура, содержащая информацию о процессе в следующих полях:

`Iterations` – число итераций,

`funcCount` – значение целевой функции, `firstorderopt` – условие оптимальности 1-го порядка, `algorithm` – использованный алгоритм,

`cgiterations` – общее число итераций (для алгоритма большой размерности),

`stepsize` – заключительное смещение по X (для алгоритмов средней размерности),

`message` – сообщение о завершении.

Входные аргументы:

`fun` – либо только целевая функция, либо функция, возвращающая значения целевой функции и ее градиента, либо функция, возвращающая значения целевой функции, ее градиента и гессиана; в первом случае она может задаваться, как показано в гл. 1, а во втором случае – *m*-файлом с двумя выходными аргументами f и g, причем градиент g – векторный аргумент (элементы разделяются точкой с запятой). Пример:

```
function [f,g] = myfun1(x) f = 2*x(1)^2+x(2)*x(1)+x(2)^3;
g =[4*x(1)+x(2);x(1)+3*x(2)^2];
```

в третьем случае – также *m*-файлом с тремя выходными аргументами:

```
function [f,g,H] = myfun2(x)f = ...;
g = ...;
H = ...;,
```

где `H` – матрица вторых производных целевой функции.

Чтобы аналитические выражения градиента и гессиана использовались в алгоритме, их необходимо включить в `options`:

```
options=optimset('GradObj','on')
```

или `options=optimset('GradObj','on',’Hessian’,’on’)`.

В противном случае они будут вычисляться приближенно через конечные разности;

`x0` – начальная точка, задается пользователем;

`options` – используется для внесения изменений в настройки параметров процесса;

`problem` – задает задачу указанием `objective` – целевая функция, `x0` – начальная точка,

`solver` –`'fminunc'`

`options` – настройки параметров.

Следует иметь в виду, что в MATLAB выходные аргументы, как и входные, могут указываться только в том порядке, который предписан синтаксисом обращения к функции минимизации. Так, например, при обращении к `fminunc` недопустима запись

```
[x,fval,output] = fminunc(...)
```

из-за пропуска аргумента `exitflag`.

Пример 1.

Рассмотрим тот же пример, что рассматривали в гл. 1. Здесь мы будем учитывать как A1, так и A2. Однако мы больше не сможем учитывать ограничения 2-го рода.

Тогда, заменив A1 на X1, получим следующее формализованное описание задачи оптимизации:

$$\begin{cases} C(X1,X2) = \alpha * (X1^2 + \beta * X2 - \mu * V1)^N + \alpha1 * (\beta1 * X1 + X2^2 - \mu1 * V2)^N \rightarrow ma \\ \alpha = \alpha1 = \beta = \beta1 = \mu = \mu1 = 1 \\ N = 2 \\ V1 = 11 \\ V2 = 7 \end{cases}$$

Шаг 1. Отредактируем целевую функцию. Для этого отредактируем код функции «TagretFunc»:

```
function C = TargetFunc(a,A1,B,A2,u,V1,N,a1,B1,u1,V2)
C=a*(A1.^2+B*A2-u*V1).^N+a1*(B1*A1+A2.^2-u1*V2).^N;
end
```

Шаг 2. Отредактируем функцию построения графика целевой функции «FuncPlot». Теперь это будет функция построения графика линий равных значений:

```
function FuncPlot(a,B,X1l,X1u,X2l,X2u,u,V1,N,a1,B1,u1,V2)
x=linspace(X1l, X1u, 100);
y=linspace(X2l, X2u, 100);
[X,Y]=meshgrid(x,y);
Z=TargetFunc(a,X,B,Y,u,V1,N,a1,B1,u1,V2);
contour(X,Y,Z,'ShowText', 'on');
xlabel('x');
ylabel('y');
title('Maximization by fminunc');
hold on;
end
```

Функция linspace – создаёт массив от X1l или X2l до X1u или X2u количеством 100. Функция meshgrid создаёт представление двух массивов для последующей функции построения графика. Z – массив значений функции. Функция «contour» - функция построения графика линий равных значений.

Шаг 3. Изменим функцию запуска процесса оптимизации «Optim»:

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,B,X1l,X2l,u,V1,N,a1,B1,u1,V2)
fValuesSolution=[];
xValuesSolution=[];
options=optimset('outputfcn', @outfun, 'TolX', 0.1);
[x,fval,exitflag,output] = fminunc(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2), [X1l,X2l], options);
```

В аргументах вместо нижней и верхней границ одного из варьируемых параметров теперь присутствует нижние границы варьируемых аргументов как начальная точка. Теперь `@(X)` – массив из значений всех переменных, поэтому `X(1), X(2)` – обращения к элементам 1 и 2 точки.

Шаг 3. Изменим начальный скрипт:

```
a=1;B=1;u=1;V1=11;N=2;a1=1;B1=1;u1=1;V2=7;
A1l=1;A1u=10;A2l=1;A2u=10;
FuncPlot(a,B,A1l,A1u,A2l,A2u,u,V1,N,a1,B1,u1,V2);
```

```
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,B,A1l,A2l,u,V1,N,a1,B1,u1,V2);
AddSolutionToPlot(xValuesSolution(:,1),xValuesSolution(:,2));
exitflag
output
x
F
```

Мы добавили нижнюю и верхнюю границы первого варьируемого параметра, передали в функцию построения траектории движения точки, по которым «ходил» метод оптимизации.

Листинг кода представлен ниже:

```
a=1;B=1;u=1;V1=11;N=2;a1=1;B1=1;u1=1;V2=7;
A1l=1;A1u=10;A2l=1;A2u=10;
FuncPlot(a,B,A1l,A1u,A2l,A2u,u,V1,N,a1,B1,u1,V2);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,B,A1l,A2l,u,V1,N,a1,B1,u1,V2);
AddSolutionToPlot(xValuesSolution(:,1),xValuesSolution(:,2));
exitflag
output
x
F
```

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,B,X1l,X2l,u,V1,N,a1,B1,u1,V2)
fValuesSolution=[];
xValuesSolution=[];
options=optimset('outputfcn', @outfun, 'TolX', 0.1);
[x,fval,exitflag,output] = fminunc(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2), [X1l,X2l], options);
F=-fval;
    function stop = outfun(x, optimValues, state)
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -
optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
end
```

```
function C = TargetFunc(a,A1,B,A2,u,V1,N,a1,B1,u1,V2)
C=a*(A1.^2+B*A2-u*V1).^N+a1*(B1*A1+A2.^2-u1*V2).^N;
end


function FuncPlot(a,B,X1l,X1u,X2l,X2u,u,V1,N,a1,B1,u1,V2)
x=linspace(X1l, X1u, 100);
y=linspace(X2l, X2u, 100);
[X,Y]=meshgrid(x,y);
Z=TargetFunc(a,X,B,Y,u,V1,N,a1,B1,u1,V2);
contour(X,Y,Z,'ShowText', 'on');
xlabel('x');
ylabel('y');
title('Maximization by fminunc');
hold on;
end


function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
end
```

Одного взгляда на результаты достаточно, чтобы понять, что этот метод оптимизации не подходит для этой целевой функции, он даже не приблизился к максимуму функции и вышел за, пусть и не поставленную в задаче, нижние границы варьируемых параметров.

```
Local minimum possible.

fminunc stopped because the size of the current step is less
than
the value of the step size tolerance.

<stopping criteria details>
```

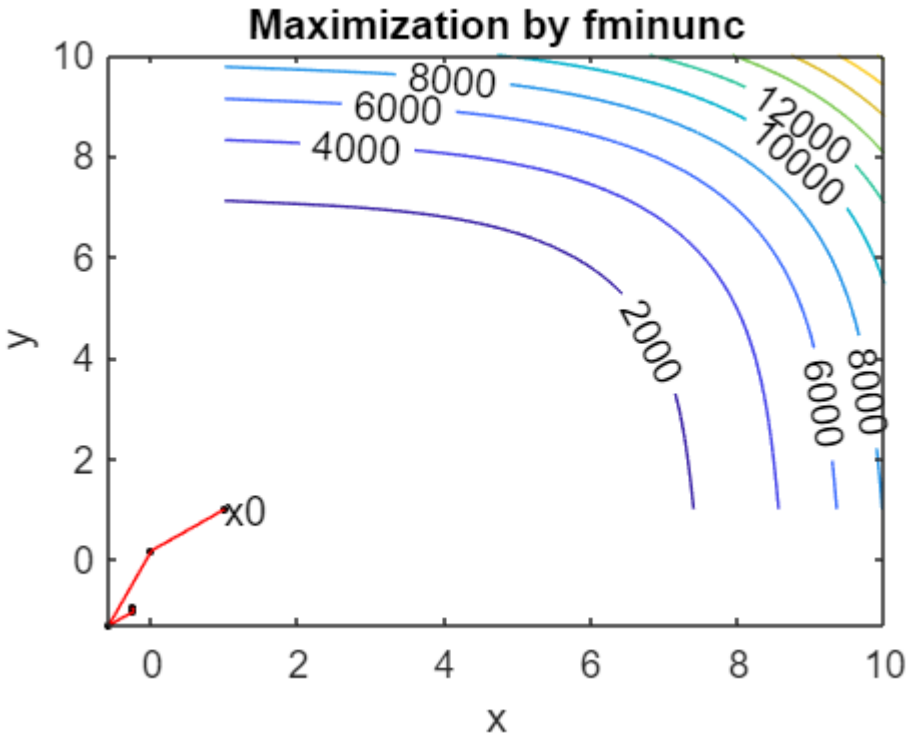

Рисунок 5 – Траектория метода оптимизации

```
exitflag = 2
output = struct with fields:
       iterations: 4
        funcCount: 15
         stepsize: 0.0816
     lssteplength: 1
    firstorderopt: 0.4766
        algorithm: 'quasi-newton'
          message: '↲Local minimum possible.↲↲fminunc stopped
because the size of the current step is less than↲the value of
the step size tolerance.↲↲<stopping criteria
details>↲↲Optimization stopped because the norm of the current
step, 4.114746e-02, is↲less than options.StepTolerance =
1.000000e-01.↲↲'
x = 1×2
   -0.2573   -0.9504
F = 181.6079
```

Пример 2.

Поставим задачу оптимизации из 2-о примера из гл. 1. Как и в предыдущем примере учтём ограничения метода оптимизации и сформулируем соответствующую задачу оптимизации:

$$\begin{cases} V(X1, X2) = a * (X1 - \beta * \Delta p1) * \cos(\gamma * \Delta p2 * \sqrt{X1^N + X2^N}) \rightarrow max \\ \Delta p1 = \Delta p2 = \alpha = \alpha 1 = \beta = 1 \\ \gamma = 3{,}14 \\ N = 2 \end{cases}$$

При замене целевой делаем то же, что и в гл. 1. Однако в этому случае в функции «FuncPlot» в функция «linspace» для x и y изменить количество итераций с 100 на 2.

Листинг скрипта представлен ниже:

```
p1=1;p2=1;a=1;a1=1;B=1;y=3.14;N=2;
T1l=-3;T1u=0;T2l=-0.5;T2u=3;
FuncPlot(a,T1l,T1u,B,p1,y,p2,T2l,T2u,N);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,T1l,B,p1,y,p2,T2l,N);
AddSolutionToPlot(xValuesSolution, fValuesSolution);
exitflag
output
x
F
```

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output]
= Optim(a,X1l,B,p1,y,p2,X2l,N)
fValuesSolution=[];
xValuesSolution=[];
options=optimset('outputfcn', @outfun, 'TolX', 0.01);
[x,fval,exitflag,output] = fminunc(@(T)-
TargetFunc(a,T(1),B,p1,y,p2,T(2),N), [X1l, X2l], options);
F=-fval;
    function stop = outfun(x, optimValues, state)
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -
optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
```

```
end


function V = TargetFunc(a,T1,B,p1,y,p2,T2,N)
V=a*(T1-B*p1)*cos(y*p2*sqrt(T1^N+T2.^N));
end


function FuncPlot(a,T1l,T1u,B,p1,y,p2,T2l,T2u,N)
x=linspace(T1l, T1u, 2);
y1=linspace(T2l, T2u, 2);
[X,Y]=meshgrid(x,y1);
Z=TargetFunc(a,X,B,p1,y,p2,Y,N);
contour(X,Y,Z,'ShowText', 'on');
xlabel('x');
ylabel('y');
title('Maximization by fminunc');
end


function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
end
```

Результаты также показывают, что метод оптимизации не подходит для этой целевой функции.

```
Local minimum possible.

fminunc stopped because the size of the current step is less
than
the value of the step size tolerance.

<stopping criteria details>
```

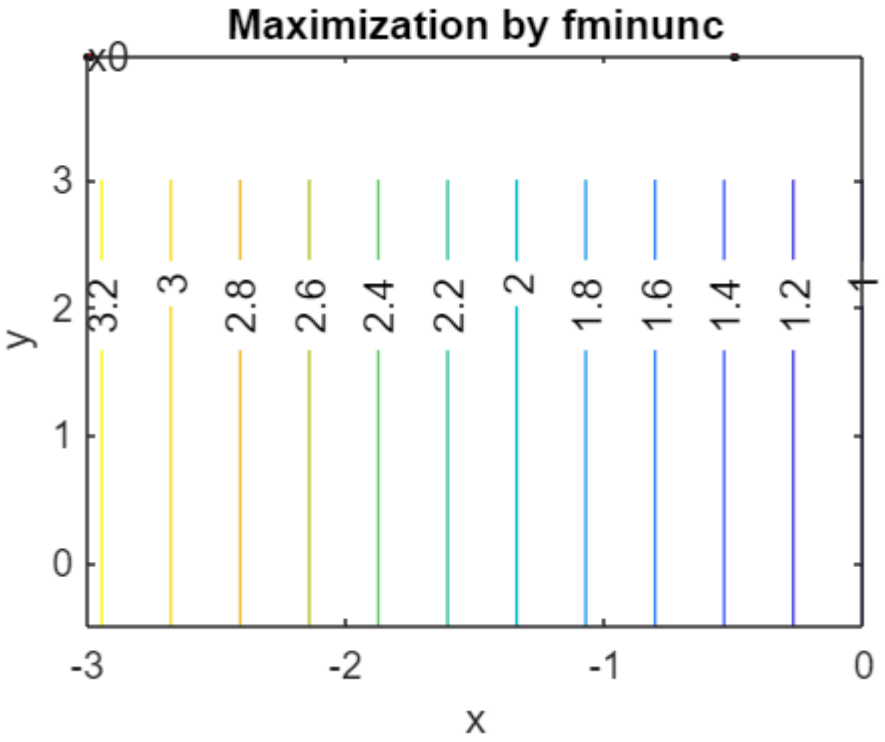

Рисунок 6 – Траектория движения метода оптимизации

```
exitflag = 2
output = struct with fields:
       iterations: 1
        funcCount: 12
         stepsize: 0.0165
     lssteplength: 0.0270
    firstorderopt: 0.1531
        algorithm: 'quasi-newton'
          message: '↲Local minimum possible.↲↲fminunc stopped
because the size of the current step is less than↲the value of
the step size tolerance.↲↲<stopping criteria
details>↲↲Optimization stopped because the norm of the current
step, 4.669298e-03, is↲less than options.StepTolerance =
1.000000e-02.↲↲'
x = 1×2
   -2.9850   -0.4930
F = 3.9738
```

Функция `fminsearch`

В функции `fminsearch` для нахождения минимума используется симплексный метод прямого поиска Нелдера–Мида. Изменение параметров

производится посредством функции optimset. Параметры fminsearch Display, FunValCheck, MaxFunEvals, MaxIter, OutputFcn, PlotFcns, TolFun и TolX идентичны одноименным параметрам для функции fminunc. Отличие только в меньшем числе функций графики: из перечисленных для fminunc в PlotFcns используются первые три.

Обращение к функции возможно в следующих вариантах:

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options) x = fminsearch(problem) [x,fval] = fminsearch(...)
[x,fval,exitflag] = fminsearch(...)
[x,fval,exitflag,output] = fminsearch(...)
```

Смысл входных и выходных аргументов такой же, как в функции fminunc. Отличие в следующем. Fun может содержать только целевую функцию, и, соответственно, в выходных аргументах нет grad и Hessian, а exitflag может принимать только следующие значения:

1 – получено решение в результате сходимости процесса,

0 – число итераций превысило допустимое значение,

−1 – процесс остановлен функцией вывода (the output function).

Пример 1.

Задача оптимизации и её постановка не отличаются от примера 1 в изучении метода оптимизации «fminunc». Меняется одна строка в функции «Optim» с

```
[x,fval,exitflag,output] = fminunc(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2), [X1l,X2l], options);
```

на

```
[x,fval,exitflag,output] = fminsearch(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2), [1.3,1.3], options);
```

По результатам видно, что метод оптимизации не подходит для максимизации целевой функции.

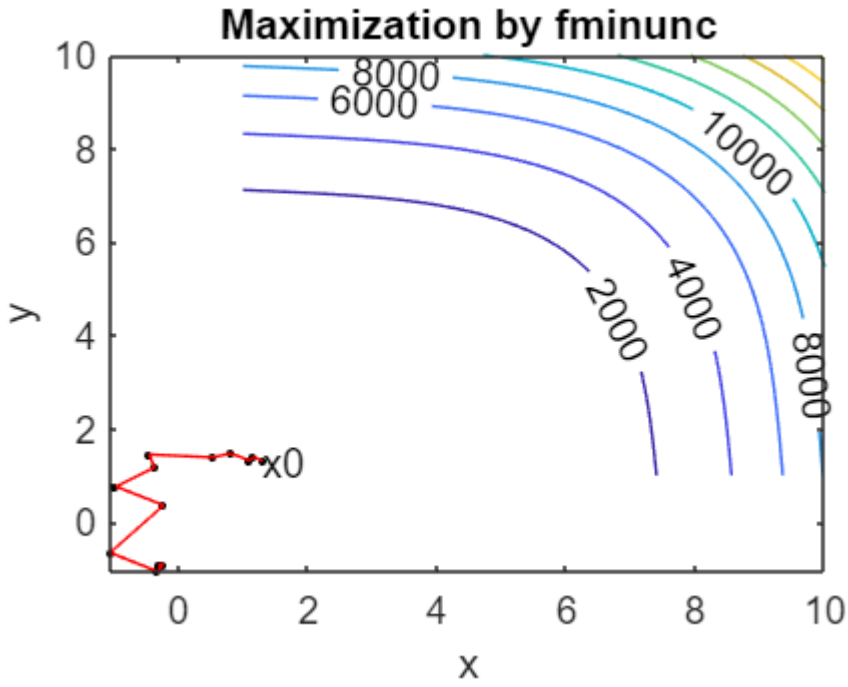

Рисунок 7 – Траектория движения метода оптимизации

```
exitflag = 1
output = struct with fields:
    iterations: 33
     funcCount: 63
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x
satisfies the termination criteria using OPTIONS.TolX of
1.000000e-01 ↵ and F(X) satisfies the convergence criteria using
OPTIONS.TolFun of 1.000000e-04 ↵'
x = 1×2
   -0.2697   -0.9239
F = 181.6165
```

Пример 2.

Задача оптимизации и её постановка не отличаются от примера 2 в изучении метода оптимизации «fminunc». Меняется одна строка в функции «Optim» с

```
[x,fval,exitflag,output] = fminunc(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2), [X1l,X2l], options);
```
на
```
[x,fval,exitflag,output] = fminsearch(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2), [1.3,1.3], options);
```

По результатам видно, что метод оптимизации не подходит для максимизации и этой целевой функции.

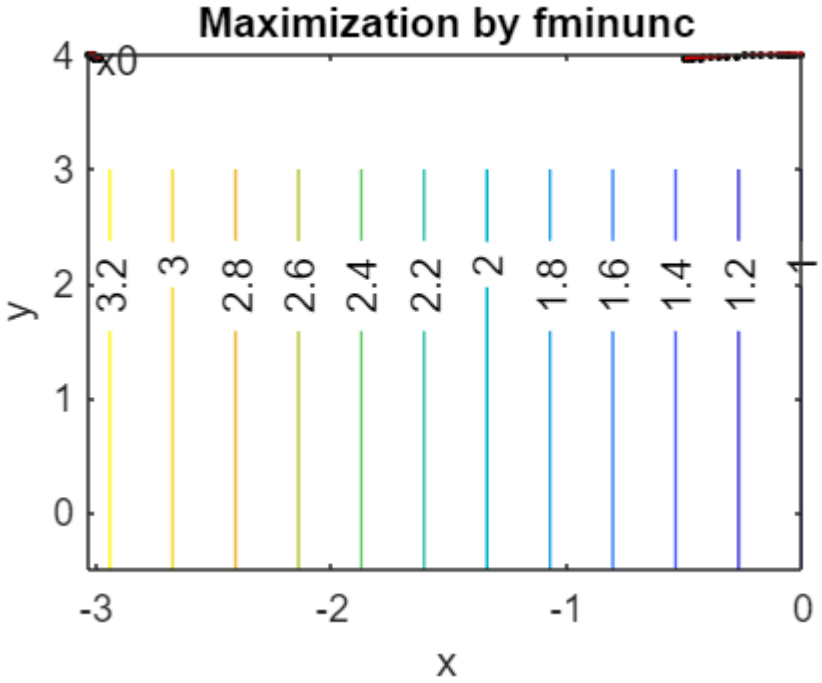

Рисунок 8 – Траектория движения метода оптимизации

```
exitflag = 1
output = struct with fields:
    iterations: 34
     funcCount: 67
     algorithm: 'Nelder-Mead simplex direct search'
       message: 'Optimization terminated:↵ the current x
satisfies the termination criteria using OPTIONS.TolX of
1.000000e-02 ↵ and F(X) satisfies the convergence criteria using
OPTIONS.TolFun of 1.000000e-04 ↵'
x = 1×2
   -3.0265    0.0019
F = 4.0141
```

УСЛОВНАЯ ОПТИМИЗАЦИЯ

Для поиска минимума функции многих переменных при наличии ограничений предназначена функция `fmincon` из пакета `Toolbox`. Фактически с помощью этой функции можно решать задачи нелинейного математического программирования (НМП) в самой общей постановке. В обозначениях MATLAB задача НМП ставится так:

$$\min f(X)$$

при условиях

$$C(X) \le 0,\ ceq(X) = 0,$$

$$\boldsymbol{A}\cdot X \le b,\ \boldsymbol{Aeq}\cdot X = beq, lb \le X \le ub,$$

где X – вектор переменных: $C(X)$ и $ceq(X)$ – вектор-функции левых частей нелинейных неравенств и равенств; $\boldsymbol{A}$ и $\boldsymbol{Aeq}$ – матрицы условий линейных неравенств и равенств; b и beq – векторы свободных членов (правых частей) линейных неравенств и равенств; lb и ub – векторы нижних и верхних границ на переменные. Очевидно, что в частных постановках задач НМП некоторые из представленных условий могут отсутствовать.

В связи с многообразием задач НМП функция `fmincon` реализует не один, а четыре метода условной минимизации. Рассмотрим кратко эти методы.

***Метод доверительных областей* (`Trust Region`).** Доверительной областью названа окрестность N текущей точки поиска, размеры которой могут изменяться в зависимости от поведения целевой функции. В этой области целевая функция $f(x)$ аппроксимируется болеепростой функцией $q(s)$, затем решается подзадача, заключающаяся в поиске минимума функции $q(s)$ на N, в результате находится пробный шаг s. Если окажется, что $f(x + s) < f(x)$, текущая точка изменяется и становится равной $x + s$, в ином

случае текущая точка не изменяется, доверительная область уменьшается и заново решается подзадача (находится новый пробный шаг).

В стандартном методе доверительных областей используется квадратичная аппроксимация, определяемая членами только первого и второго порядка ряда Тейлора для $f(x)$, а в качестве N – сфера или эллипсоид. Для нахождения минимума квадратичной функции (решения подзадачи) существуют хорошие алгоритмы (ньютоновские и др.), однако прибольшой размерности они требуют слишком много времени. Поэтому подход, примененный в `Toolbox Optimization`, состоит в ограничении подзадачи двумерным подпространством S. Очевидно, что в таком случае решение подзадачи становится тривиальным. Решатель определяет S как линейное пространство, натянутое на векторы s_1 и s_2, где s_1 –направление градиента G аппроксимируемой функции, а s_2 – направление, определяемое или как приближенное направление Ньютона из решения линейной системы

$$Hs_2 = -G,$$

в которой H – матрица Гессе, или как направление отрицательной кривизны поверхности целевой функции из неравенства

$$s^T Hs < 0.$$

В первом случае решение системы равенств находится предобусловленным методом сопряженных градиентов `PCG` (`PreconditionedConjugate Gradient Method`).

Таким образом, при решении задачи без ограничений основными шагами алгоритма будут:

формулировка двумерной подзадачи доверительной области;

решение подзадачи и определение s;

если $f(x + s) < f(x)$, тогда $x = x + s$;

корректировка доверительной области.

Шаги повторяются до установления сходимости. Задача усложняется при наличии ограничений. Если ограничения описываются линейной

системой равенств, метод на каждой итерации генерирует допустимое решение. Для получения допустимой начальной точки применяется шаг метода наименьших квадратов, на последующих шагах линейные системы решаются упрощенным методом сопряженных градиентов (`PCG` заменен на `Reduce PCG`). При двухсторонних ограничениях на переменные также генерируются только строго допустимые решения, для чего используются соответствующие необходимые условия Куна – Таккера.

Метод доверительных областей эффективен для решения больших разреженных задач или средних плотных задач с двухсторонними ограничениями на переменные или только с линейными ограничениями. Алгоритм требует задания градиента целевой функции и указания на его включение в `optimset`.

Метод активного набора (`Active Set`). Сегодня методы, основанные на решении уравнений Каруша – Куна – Таккера (ККТ), считают более эффективными, чем методы типа штрафных функций. Для задач выпуклого НМП условия ККТ являются одновременно необходимыми и достаточными для точки глобального решения.

Для общей задачи НМП, представляемой как

$$\min f(X)$$

$$G_i(x) = 0, \ i = 1, ..., m_e,$$

$$G_i(x) \le 0, \ i = m_e + 1, ..., m,$$

условия Куна – Таккера записываются в следующем виде:

$$\nabla f(x^*) + \sum_i^m \lambda \nabla G_i(x^*) = 0,$$

$$\lambda_i G_i(x^*) = 0, \overline{i = 1, m},$$

$$\lambda_i \geq 0, \quad i = m_e + 1, m.$$

Первое уравнение описывает взаимосвязь градиента целевой функциис градиентами активных ограничений: первый уравновешивается градиентами ограничений с весами λ_i – множителями Лагранжа. Множители неактивных ограничений равны нулю, что выражается вторым уравнением; положительными могут быть только множители активныхограничений.

Решение уравнений ККТ составляет основу многих алгоритмов нелинейного программирования. Используемые для их решения квазиньютоновские методы с процедурой обновления обеспечивают сверхлинейную скорость сходимости. Основная идея метода активного набора состоит в квадратичной аппроксимации функции Лагранжа. В таком варианте метод часто упоминается как последовательное квадратичное программирование (SQP). На каждой итерации методом квадратичного программирования решается квадратичная подзадача, определяющая направление спуска, а шаг в этом направлении находится методом линейного поиска минимума штрафной функции. Для обновления матрицы, аппроксимирующей гессиан функции Лагранжа, и обеспечения ее положительной определенности применяется метод BFGS. Алгоритм активного набора не предназначен для больших задач. Он эффективен для некоторых задач средней размерности с негладкими ограничениями. Для повышения скорости он может делать большие шаги.

Алгоритм последовательного квадратичного программирования. Он подобен методу активного набора, однако имеет важные отличия. Во-первых, SQP соблюдает строгую допустимость относительно заданных границ. На каждой итерации делается шаг в области, определяемой границами. И завершающие изменения шагов также соблюдают границы. При нестрогих границах может быть сделан шаг точно на границу. Такая строгая допустимость выгодна, когда целевая функция или функции нелинейных ограничений неопределенны или сложны вне границ. Во-

вторых, алгоритм обеспечивает робастность движения: при неудачном шаге, когда функция или функции задачи оказываются неопределенными (они возвращают значения `Inf, NaN`), алгоритм пытается уменьшить шаг. В-третьих, `SQP` отличается от алгоритма `Active Set` использованием более эффективных программ линейной алгебры, которые вызываются при решении квадратичной подзадачи. Наконец, в `SQP` реализованы два новых подхода к решению подзадачи квадратичного программирования в случае невыполнения ограничений. Алгоритм использует функцию качества, которая образуется комбинацией целевой функции и ограничений. Минимизация функции качества приводит к допустимому решению, однако из-за увеличения числа переменных может замедлиться решение подзадачи. Второй подход к устранению нарушения ограничений связан с квадратичной аппроксимацией функцийограничений. Он также позволяет достичь допустимого решения, но требует большего числа вычислений функций ограничений, что приводит к замедлению процесса решения подзадачи квадратичного программирования.

Алгоритм `SQP` применяется для решения задач средней размерности.

Метод внутренней точки (`Interior-point`). Суть метода заключается в решении последовательности аппроксимирующих задач. Для общей задачи НМП

$$\min_x f(x)$$

при условиях

$$h(x) = 0, \quad g(x) \le 0$$

аппроксимирующая задача для каждого $\mu > 0$ имеет вид

$$\min_{x,s} f_\mu(x,s) = \min_{x,s}(f(x) - \mu \sum_i \ln(s_i))$$

при условиях

$$h(x) = 0, \qquad g(x) + s = 0.$$

Каждому ограничению-неравенству исходной задачи соответствует положительная переменная s_i в аппроксимирующей задаче. При стремлении μ к нулю минимум f_μ должен приближаться к минимуму *f*. Добавленный к целевой функции член называется *барьерной функцией*.Как видно, аппроксимирующая задача представляет собой последовательность задач с ограничениями-равенствами, и ее решение найтилегче, чем исходной задачи с неравенствами.

На каждой итерации используется один из двух основных типов шагов:

прямой шаг в пространстве (*x*, *s*) для решения ККТ уравнений аппроксимирующей задачи путем их линейной аппроксимации; такой шаг называется шагом Ньютона;

CG-шаг (conjugate gradient – сопряженный градиент) с использованием доверительной области.

По умолчанию сначала делается попытка выполнить прямой шаг. Если он невозможен, алгоритм пытается сделать CG-шаг. Неудача пря мого шага может быть обусловлена локальной невыпуклостью решаемой задачи вблизи текущей точки. На каждой итерации алгоритм уменьшает функцию качества, например такую, как

$$f_\mu(x,s) + \nu||(h(x), g(x)+s)||.$$

Параметр ν может увеличиваться по ходу итераций для продвижения решения к допустимой области. Если предпринятый шаг не уменьшает функцию качества, алгоритм отклоняет его и пытается сделать новый шаг. Если в точке x_j функции возвращают комплексное значение, `NaN`, `Inf` или ошибку, x_j отвергается (как в случае недостаточного уменьшения функции качества) и делается другой, более короткий шаг. В начальной точке подобное недопустимо.

Метод внутренней точки применяется для решения задач как большой, так и средней размерности. Если в опциях не указан конкретный алгоритм и применение алгоритма по умолчанию, т.е. `TrustRegion`, к решаемой задаче невозможно, программа обращается к алгоритму внутренней точки.

Процесс оптимизации определяется не только выбранным алго- ритмом, но и значениями параметров. Изменение параметров осущест- вляется с помощью функции `optimset`. Часть параметров относитсяко всем алгоритмам, и каждый алгоритм имеет еще свои индивидуаль- ные параметры. Все параметры `fmincon` и их описание приведены в прил. 2.

Одним из существенных параметров является `'Hessian'`, уста- навливающий способ вычисления гессиана. Все алгоритмы, исключая `SQP`, используют гессиан функции Лагранжа

$$\nabla^2 L(x,\lambda) = \nabla^2_i f(x) + \sum \lambda \nabla^2 C(x) + \sum \lambda \nabla^2 ceq(x).$$

Алгоритмы вычисляют гессиан по-разному. Алгоритм `active-set` не использует пользовательский гессиан, а вычисляет его методом квази-ньютоновской аппроксимации. Алгоритм trust-region-reflective может принимать гессиан, заданный пользователем. Поскольку этот алгоритм не предназначен для нелинейных ограничений, гессиан функции Лагранжа совпадает с гессианом целевой функции и описывается вместе с ней. Алгоритм interior-point воспринимает гессиан, заданный пользователем в виде отдельной функции, описание которой имеет вид

`hessian = hessianfcn(x, lambda)` hessianfcn=гессиан Лагранжа,

где `hessian` квадратная матрица n×n, `lambda` – множители Лагранжа, соответствующие нелинейным ограничениям. Пользовательская функция указывается в `optimset`:

`options = optimset('Hessian','user-supplied',... 'HessFcn',@hessianfcn).`

В алгоритме `interior-point` можно выбрать способ вычисления гессиана из пяти вариантов, задаваемых как значение параметра `'Hessian'`: `'bfgs'` – посредством квази-ньютоновской аппроксимации; `{'lbfgs',positive integer})`– то же, но с ограниченной па- мятью на сохранение числа прошедших итераций (целое число); `'lbfgs'`– то же, но память на 10 итераций; `'fin-diff-grads'`– конечными разностями градиентов, при этом все градиенты задаютсяаналитически; `'user-supplied'`– способ задает пользователь.

Теперь рассмотрим способы использования функции `fmincon`, ее входные аргументы и возвращаемые величины (выходные аргументы).

Обращение к функции `fmincon` записывается в одном из следующих возможных вариантов:

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)x =
fmincon(problem)
[x,fval] = fmincon(...) [x,fval,exitflag] =
fmincon(...) [x,fval,exitflag,output] = fmincon(...)
[x,fval,exitflag,output,lambda] = fmincon(...)
[x,fval,exitflag,output,lambda,grad] = fmincon(...)
[x,fval,exitflag,output,lambda,grad,hessian] = ...fmincon(...)
```

Смысл большинства *входных аргументов* ясен из описания задачи НМП и функции `fminunc`. В частности, `fun` задается функцией, кото- рая может возвращать значение как целевой функции, так и градиентаи гессиана, если это необходимо. Отличие только в

присутствии ново- го аргумента `nonlcon`, который представляет нелинейные условия задачи в виде *m*-файла, например

```
function [c,ceq] = mycon(x)c = ...;
ceq = ...;
```

В этом случае в качестве `nonlcon` следует записать `@mycon`. Если ра- венств и неравенств больше одного, то `c` и `ceq` записываются как век-торы. При отсутствии равенств или неравенств в правой части соответ-ствующего выражения записывается `[]`. В MATLAB при записи мат- рицы (вектора) элементы строки разделяются пробелом или запятой, а строки отделяются точкой с запятой или записываются в разных стро- ках матрицы (вектора). Так, неравенства

$$\begin{matrix} 1 & 2 \end{matrix} \quad \begin{matrix} 2x^3 - x \le 5, \\ x + x^2 \le 12 \\ 1 \quad 2 \end{matrix}$$

в *m*-файле будут представлены в виде

```
c =[2*x(1)^3-x(2)-5;x(1)+x(2)^2-12];.
```

Если нужно указать конкретный алгоритм, то в опции (в `optim- set`) записывается параметр `'Algorithm'` со значением `'interior-point'` либо `'sqp'`, либо `'active-set'`. По умолчанию применяетсяалгоритм `'trust-region-reflective'` и, если он не подходит (не задан градиент целевой функции или не соответствует тип ограничений), программа заменяет его на `interior-point`.

Выходные аргументы `fmincon` отличаются от аргументов `fmin- unc` значениями `exitflag`, структурой `output` и новым аргументом `lambda`. Значение `exitflag` указывает на причину завершения алгоритма:

а) для всех алгоритмов

`1` – мера оптимальности первого порядка стала меньше установленной в `options.TolFun` и максимальное нарушение ограничений меньше `options.TolCon` (получено решение с заданной точностью),

`0` – превышено максимальное число итераций `(options.Max- Iter)` или вычислений функции (`options.FunEvals`),

`-1` – алгоритм завершен функцией `output`,

`-2` – не найдено допустимой точки;

б) для алгоритмов доверительных областей и внутренней точки

– изменение в *X* меньше установленного в `options.TolX` и максимальное нарушение ограничений меньше `options.TolCon` (получено решение с заданной точностью);

в) только для алгоритма доверительных областей

– изменение целевой функции меньше options.TolFun и максимальное нарушение ограничений меньше options.TolCon (получено решение с заданной точностью);

г) только для алгоритма активного набора

– величина направления поиска меньше 2*options.TolX и максимальное нарушение ограничений меньше options.TolCon (получено решение с заданной точностью),

– величина производной по направлению прямого поиска мень- ше 2*options.TolFun и максимальное нарушение ограничений меньше options.TolCon (получено решение с заданной точностью);

д) только для алгоритма внутренней точки

-3 – текущая точка X ниже величины options.ObjectiveLimitи максимальное нарушение ограничений меньше options.TolCon (получено решение с заданной точностью).

Структура Output содержит поля iterations, funcCount, lssteplength (в алгоритме активного набора), constrviolation, stepsize (оба в алгоритмах активного набора и внутренней точки), algorithm, cgiterations (в алгоритмах доверительных областей ивнутренней точки), firstorderopt, message. Пояснения требуюттолько новые поля: lssteplength – размер шага линейного поиска относительно направления поиска, constrviolation – максимальноезначение функций ограничений (величина нарушения ограничений).

Lambda – структура, содержащая множители Лагранжа, отражающие влияние ограничений в оптимальном решении; состоит из полей, соответствующих определенным ограничениям:

Lower – нижняя граница (lb), Upper – верхняя граница (ub), Ineqlin – линейные неравенства,Eqlin – линейные равенства,

Ineqnonlin – нелинейные неравенства,

Eqnonlin – нелинейные равенства.

Рассмотрим несколько примеров применения функции fmincon.

Пример 1. Поставим задачу из примера 1 из предыдущих глав. На этот раз формализованная версия будет самой полной.

$$\begin{cases} C(X1,X2) = \alpha * (X1^2 + \beta * X2 - \mu * V1)^N + \alpha1 * (\beta1 * X1 + X2^2 - \mu1 * V2)^N \to max \\ \alpha = \alpha1 = \beta = \beta1 = \mu = \mu1 = 1 \\ N = 2 \\ V1 = 11 \\ V2 = 7 \\ 1 \le A1 \le 10 \\ 1 \le A2 \le 10 \\ A1 + A2 \le 8 \end{cases}$$

Шаг 1. Добавим функцию «confun», отражающую ограничения 2-го рода:

```
function [c,ceq] = confun(X)
c=X(1)+X(2)-8;
ceq=[];
end
```

Здесь c – уравнение с перенесённой из неравенства правой частью. При это неравенство должно быть ≤ 0.

Шаг 2. Изменим функцию «Optim»:

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,B,X1l,X1u,X2l,X2u,u,V1,N,a1,B1,u1,V2)
fValuesSolution=[];
xValuesSolution=[];
options=optimset('outputfcn', @outfun, 'TolX', 0.1, 'Algorithm', 'active-set');
[x,fval,exitflag,output] = fmincon(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2),...
    [(X1l+X1u)/2 (X2l+X2u)/2],[],[],[],[], [X1l, X2l],[X1u, X2u], @confun,
options);
```

Здесь мы добавили верхние границы варьируемых параметров в аргументы функции, а также в метод оптимизации ([X1l, X2l],[X1u, X2u]) и использовали границы для вычисления начальной точки ([(X1l+X1u)/2 (X2l+X2u)/2]). Кроме того, в конце указана @confun – ограничения 2-о рода. Указан метод – активный набор.

Листинг скрипта:

```
a=1;B=1;u=1;V1=11;N=2;a1=1;B1=1;u1=1;V2=7;
A1l=1;A1u=10;A2l=1;A2u=10;
FuncPlot(a,B,A1l,A1u,A2l,A2u,u,V1,N,a1,B1,u1,V2);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] = ...
Optim(a,B,A1l,A1u,A2l,A2u,u,V1,N,a1,B1,u1,V2);
AddSolutionToPlot(xValuesSolution(:,1),xValuesSolution(:,2));
exitflag
output
x
F
```

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,B,X1l,X1u,X2l,X2u,u,V1,N,a1,B1,u1,V2)
fValuesSolution=[];
xValuesSolution=[];
options=optimset('outputfcn', @outfun, 'TolX', 0.1, 'Algorithm', 'active-set');
```

```
[x,fval,exitflag,output] = fmincon(@(X)-
TargetFunc(a,X(1),B,X(2),u,V1,N,a1,B1,u1,V2),...
    [(X1l+X1u)/2 (X2l+X2u)/2],[],[],[],[], [X1l, X2l],[X1u, X2u], @confun,
options);
F=-fval;
    function stop = outfun(x, optimValues, state)
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
end


function C = TargetFunc(a,A1,B,A2,u,V1,N,a1,B1,u1,V2)
C=a*(A1.^2+B*A2-u*V1).^N+a1*(B1*A1+A2.^2-u1*V2).^N;
end


function FuncPlot(a,B,X1l,X1u,X2l,X2u,u,V1,N,a1,B1,u1,V2)
x=linspace(X1l, X1u, 100);
y=linspace(X2l, X2u, 100);
[X,Y]=meshgrid(x,y);
Z=TargetFunc(a,X,B,Y,u,V1,N,a1,B1,u1,V2);
contour(X,Y,Z,'ShowText', 'on');
xlabel('x');
ylabel('y');
title('Maximization by fminunc');
hold on;
end


function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
end


function [c,ceq] = confun(X)
c=X(1)+X(2)-8;
ceq=[];
end
```

Результаты показали, что максимум не был найден, однако, этот метод сильнее приблизился к точке максимума, чем метод «fminsearch»

```
Active inequalities (to within options.ConstraintTolerance = 1e-06):
  lower      upper     ineqlin   ineqnonlin
    1                                1

Local minimum found that satisfies the constraints.
```

```
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the  value of the constraint
tolerance.

<stopping criteria details>
```

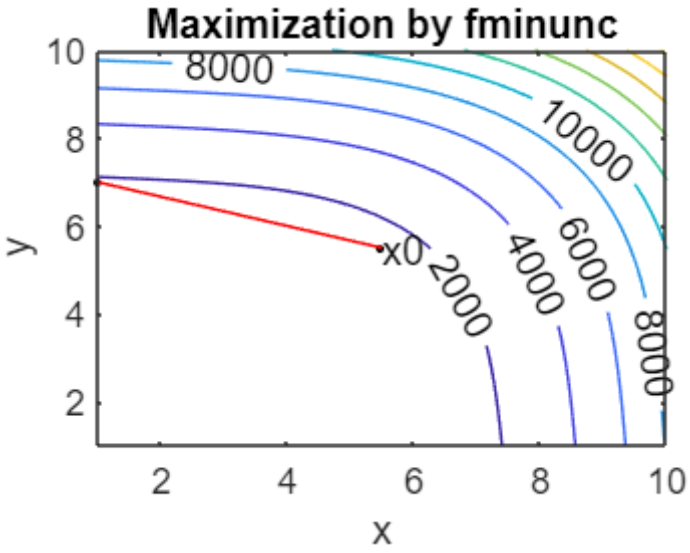

Рисунок 8 – Траектория движения метода оптимизации

```
exitflag = 1
output = struct with fields:
         iterations: 2
          funcCount: 6
       lssteplength: 1
           stepsize: 0
          algorithm: 'active-set'
      firstorderopt: 0
    constrviolation: 0
            message: '↲Local minimum found that satisfies the
constraints.↲↲Optimization completed because the objective function is non-
decreasing in ↲feasible directions, to within the value of the optimality
tolerance,↲and constraints are satisfied to within the  value of the
constraint tolerance.↲↲<stopping criteria details>↲↲Optimization completed:
The first-order optimality measure, 0.000000e+00, is less↲than
options.OptimalityTolerance = 1.000000e-06, and the maximum constraint
violation,↲0.000000e+00,is less than options.ConstraintTolerance = 1.000000e-
06.↲↲'
       bestfeasible: [1×1 struct]
x = 1×2
    1.0000    7.0000
F = 1.8580e+03
```

При указании метода «interior-point» метод так же не находит глобальный максимум, однако, ещё больше приближается к нему.

```
Converged to an infeasible point.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance but constraints are not
satisfied to within the value of the constraint tolerance.
```

<stopping criteria details>

Consider enabling the interior point method feasibility mode.

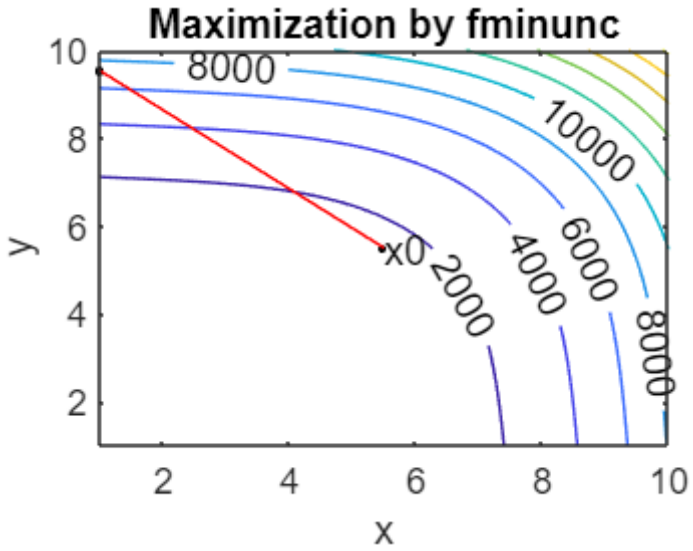

Рисунок 9 – Траектория движения метода оптимизации

```
exitflag = -2
output = struct with fields:
         iterations: 2
          funcCount: 9
    constrviolation: 2.5437
           stepsize: 0.0301
          algorithm: 'interior-point'
      firstorderopt: 1.3964e+03
       cgiterations: 0
            message: '↲Converged to an infeasible point.↲↲fmincon stopped
because the size of the current step is less than↲the value of the step size
tolerance but constraints are not↲satisfied to within the value of the
constraint tolerance.↲↲<stopping criteria details>↲↲Optimization stopped
because the relative changes in all elements of x are↲less than
options.StepTolerance = 1.000000e-01, but the relative maximum
constraint↲violation, 8.478895e-01, exceeds options.ConstraintTolerance =
1.000000e-06.↲↲'
       bestfeasible: []
x = 1×2
    1.0001    9.5436
F = 7.2387e+03
```

Попробуем метод «sqp»

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in feasible directions, to within the value of the optimality tolerance, and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>

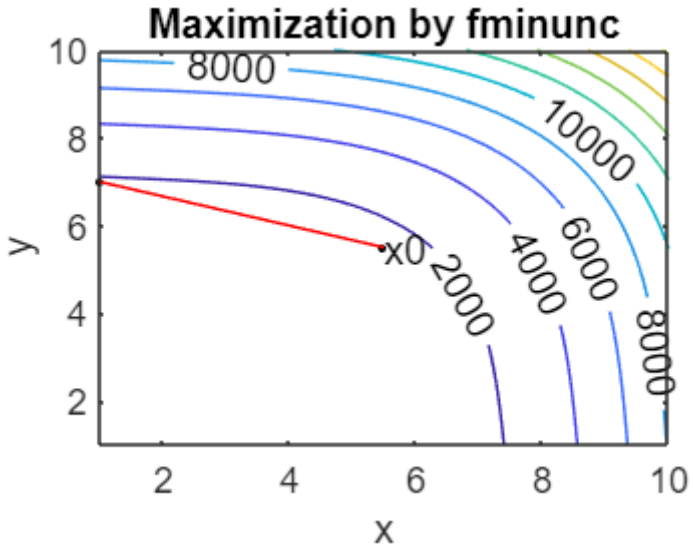

Рисунок 10 – Траектория движения метода оптимизации

```
exitflag = 1
output = struct with fields:
         iterations: 2
          funcCount: 9
          algorithm: 'sqp'
            message: '↲Local minimum found that satisfies the
constraints.↲↲Optimization completed because the objective function is non-
decreasing in ↲feasible directions, to within the value of the optimality
tolerance,↲and constraints are satisfied to within the value of the constraint
tolerance.↲↲<stopping criteria details>↲↲Optimization completed: The relative
first-order optimality measure, 1.897944e-16,↲is less than
options.OptimalityTolerance = 1.000000e-06, and the relative maximum
constraint↲violation, 6.513308e-15, is less than options.ConstraintTolerance =
1.000000e-06.↲↲'
    constrviolation: 1.9540e-14
           stepsize: 1.0000e-05
      lssteplength: 1
     firstorderopt: 2.2737e-13
      bestfeasible: [1×1 struct]
x = 1×2
    1.0000    7.0000
F = 1.8580e+03
```

Метод «sqp» находится от максимума почти так же далеко от максимума функции, как и «active-set».

Пример 2.

Формулируем задачу оптимизации:

$$\begin{cases} V(X1,X2) = a * (X1 - \beta * \Delta p1) * \cos(\gamma * \Delta p2 * \sqrt{X1^N + X2^N}) \rightarrow max \\ \Delta p1 = \Delta p2 = \alpha = \alpha 1 = \beta = 1 \\ \gamma = 3{,}14 \\ N = 2 \\ -3 \leq T1 \leq 0 \\ -0{,}5 \leq T2 \leq 3 \\ T2 - T1 \geq 3 \end{cases}$$

Всё делаем по аналогии с первым примером.

Листинг скрипта представлен ниже:

```
p1=1;p2=1;a=1;a1=1;B=1;y=3.14;N=2;
T1l=-3;T1u=0;T2l=-0.5;T2u=3;
FuncPlot(a,T1l,T1u,B,p1,y,p2,T2l,T2u,N);
[x, F, xValuesSolution, fValuesSolution, exitflag, output] = ...
Optim(a,T1l,T1u,B,p1,y,p2,T2l,T2u,N);
AddSolutionToPlot(xValuesSolution, fValuesSolution);
exitflag
output
x
F
```

```
function [x, F, xValuesSolution, fValuesSolution, exitflag, output] =
Optim(a,X1l,X1u,B,p1,y,p2,X2l,X2u,N)
fValuesSolution=[];
xValuesSolution=[];
options=optimset('outputfcn', @outfun, 'TolX', 0.01, 'Algorithm', 'active-set');
[x,fval,exitflag,output] = fmincon(@(X)-TargetFunc(a,X(1),B,p1,y,p2,X(2),N),...
    [(X1l+X1u)/2 (X2l+X2u)/2],[],[],[],[], [X1l, X2l],[X1u, X2u], @confun,
options);
F=-fval;
    function stop = outfun(x, optimValues, state)
        stop=false;
        switch state
            case 'iter'
                fValuesSolution=[fValuesSolution; -optimValues.fval];
                xValuesSolution=[xValuesSolution; x];
        end
    end
end


function V = TargetFunc(a,T1,B,p1,y,p2,T2,N)
V=a*(T1-B*p1)*cos(y*p2*sqrt(T1^N+T2.^N));
end


function FuncPlot(a,T1l,T1u,B,p1,y,p2,T2l,T2u,N)
x=linspace(T1l, T1u, 2);
y1=linspace(T2l, T2u, 2);
[X,Y]=meshgrid(x,y1);
Z=TargetFunc(a,X,B,p1,y,p2,Y,N);
contour(X,Y,Z,'ShowText', 'on');
xlabel('x');
ylabel('y');
title('Maximization by fminunc');
end


function AddSolutionToPlot(x, y)
hold on
plot(x, y,'black.', x, y, 'red-');
text(x(1), y(1), 'x0');
```

```
end


function [c,ceq] = confun(X)
c=-X(2)+X(1)+3;
ceq=[];
end
```

По представленным ниже результатам видно, что метод оптимизации не смог найти максимум целевой функции.

```
Active inequalities (to within options.ConstraintTolerance = 1e-06):
  lower      upper     ineqlin   ineqnonlin
    1

Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current search direction is less than
twice the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

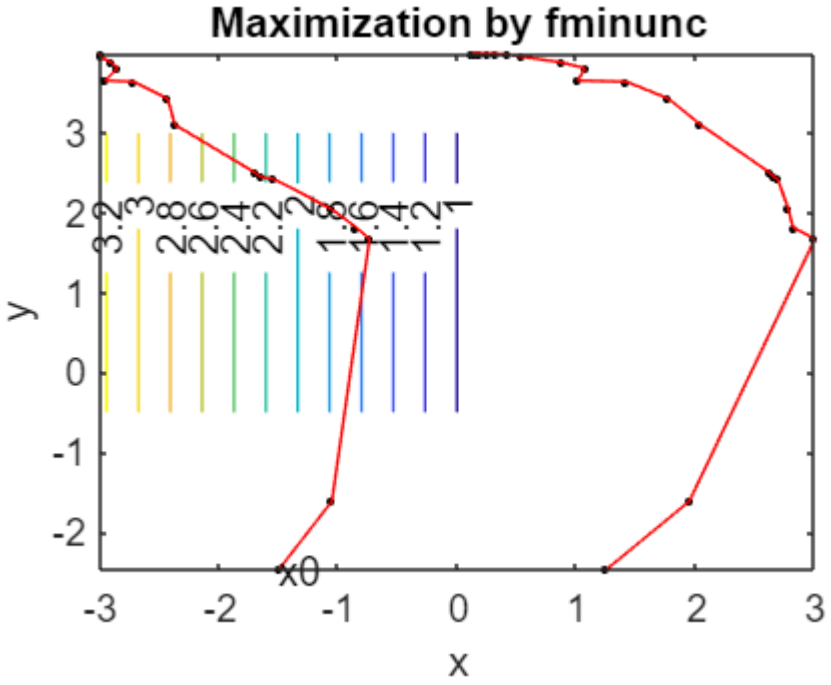

Рисунок 11 – Траектория движения метода оптимизации

```
exitflag = 4
output = struct with fields:
         iterations: 21
          funcCount: 67
       lssteplength: 1
           stepsize: 0.0166
          algorithm: 'active-set'
      firstorderopt: 0.0017
    constrviolation: 0
            message: 'Local minimum possible. Constraints satisfied.fmincon
stopped because the size of the current search direction is less thantwice
the value of the step size tolerance and constraints are satisfied to within
the value of the constraint tolerance.<stopping criteria
```

```
details>↵↵Optimization stopped because the norm of the current search
direction, 1.659382e-02,↵is less than 2*options.StepTolerance = 1.000000e-02,
and the maximum constraint ↵violation, 0.000000e+00, is less than
options.ConstraintTolerance = 1.000000e-06.↵↵'
       bestfeasible: [1×1 struct]
x = 1×2
   -3.0000    0.1246
F = 4.0000
```

Метод «interior-point»:

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

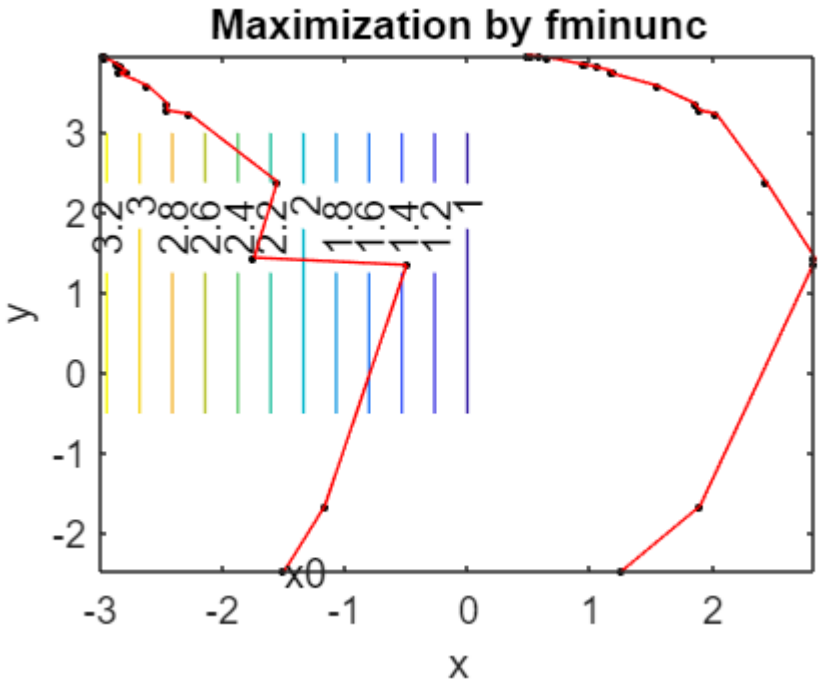

Рисунок 12 - Траектория движения метода оптимизации

```
exitflag = 2
output = struct with fields:
         iterations: 18
          funcCount: 61
    constrviolation: 0
           stepsize: 0.0045
          algorithm: 'interior-point'
      firstorderopt: 0.0199
       cgiterations: 0
            message: '↵Local minimum possible. Constraints satisfied.↵↵fmincon
stopped because the size of the current step is less than↵the value of the
step size tolerance and constraints are ↵satisfied to within the value of the
constraint tolerance.↵↵<stopping criteria details>↵↵Optimization stopped
because the relative changes in all elements of x are↵less than
options.StepTolerance = 1.000000e-02, and the relative maximum
constraint↵violation, 0.000000e+00, is less than options.ConstraintTolerance =
1.000000e-06.↵↵'
       bestfeasible: [1×1 struct]
x = 1×2
```

```
   -2.9714    0.4831
F = 3.9699
```

Метод «sqp»:

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

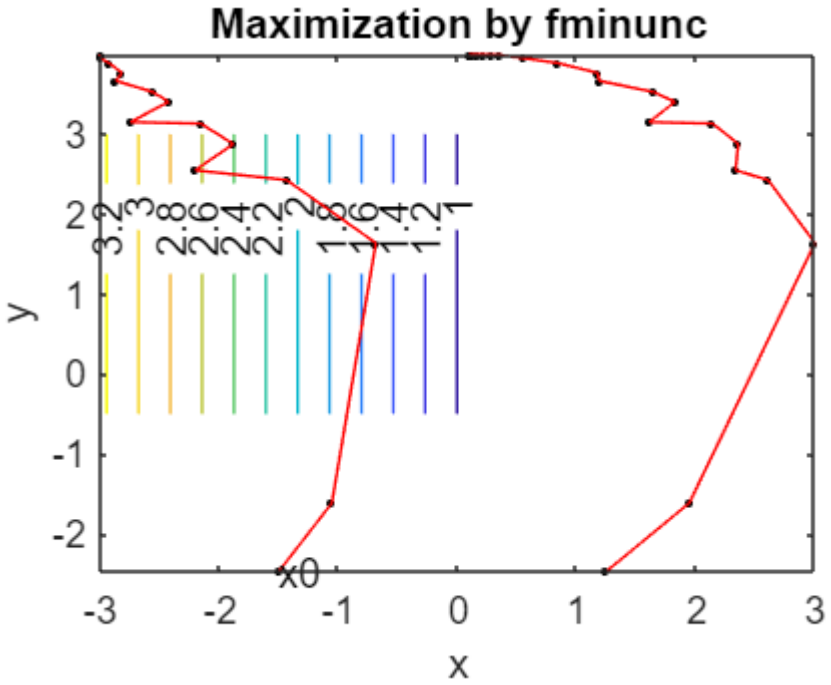

Рисунок 13 - Траектория движения метода оптимизации

```
exitflag = 2
output = struct with fields:
         iterations: 21
          funcCount: 70
          algorithm: 'sqp'
            message: 'Local minimum possible. Constraints satisfied.fmincon
stopped because the size of the current step is less thanthe value of the
step size tolerance and constraints are satisfied to within the value of the
constraint tolerance.<stopping criteria details>Optimization stopped
because the relative changes in all elements of x areless than
options.StepTolerance = 1.000000e-02, and the relative maximum
constraintviolation, 0.000000e+00, is less than options.ConstraintTolerance =
1.000000e-06.'
    constrviolation: 0
           stepsize: 0.0056
      lssteplength: 1
     firstorderopt: 6.8843e-05
      bestfeasible: [1×1 struct]
x = 1×2
   -3.0000    0.0972
F = 4.0000
```

Все методы оптимизации показали, что не могут найти максимумы представленных целевых функций.

Список использованных источников

1 Смирнов, И.А., Методы оптимизации. Контрольные работы: метод. указания / И.А. Смирнов, О.В. Ершова, Р.И. Белова – СПб.: СПбГТИ(ТУ), 2010. – 93 с.

2 Дьяконов, В. П. MATLAB. Полный самоучитель / В. П. Дьяконов. – Москва : ДМК Пресс, 2017. – 768 с. – ISBN 978-5-97060-493-9.

3 Гольдштейн А. Л. Оптимизация в среде MATLAB : учебное пособие : [для студентов (бакалавров и магистров), изучающих курс "Методы оптимизации"] / А.Л. Гольдштейн ; М-во образования и науки Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. проф. образования "Перм. нац. исслед. политехн. ун-т". - Пермь : Изд-во Пермского национального исследовательского политехнического университета, 2015. – 190 с.