Лабораторная работа № 3

УСЛОВНАЯ ОПТИМИЗАЦИЯ

Теоретические сведения

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 более простой 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 эллипсо-

ид. Для нахождения минимума квадратичной функции (решения подзадачи) существуют хорошие алгоритмы (ньютоновские и др.), однако при большой 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

$$s_2^T Hs_2 < 0.$$

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

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

1) формулировка двумерной подзадачи доверительной области;

2) решение подзадачи и определение *s*;

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

4) 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:

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

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

$$\lambda_i \ge 0, \quad i = \overline{m_e + 1, m}.$$

Первое уравнение описывает взаимосвязь градиента целевой функции с 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 активных ограничений.

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 задан- ных 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 функций ограничений. 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

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

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 найти легче, чем исходной задачи с неравенствами.

На каждой итерации используется один из двух основных типов шагов:

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

2) 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 прямого 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. `Trust`

Region, к решаемой задаче невозможно, программа обращается к алгоритму внутренней точки.

Процесс оптимизации определяется не только выбранным алгоритмом, но и значениями параметров. Изменение параметров осуществляется с помощью функции optimset. Часть параметров относится ко 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', устанавливающий способ вычисления гессиана. Все алгоритмы, исключая SQP, используют гессиан функции Лагранжа

$$\nabla^2 L(x, \lambda) = \nabla^2 f(x) + \sum \lambda_i \nabla^2 C_i(x) + \sum \lambda_i \nabla^2 ceq_i(x).$$

Алгоритмы вычисляют гессиан по-разному. Алгоритм active-set не использует пользовательский гессиан, а вычисляет его мето- дом квазиньютоновской 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 квазиньютоновской аппроксимации; {'lbfgs',positive integer})– то же, но с ограниченной памятью 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 задаются аналитически; '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, которая может возвращать значение как целевой функции, так и градиента и 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 равенств и неравенств больше одного, то `c` и `ceq` записываются как векторы. При отсутствии равенств или неравенств в правой части соответствующего выражения записывается `[]`. В MATLAB при записи матрицы (вектора) элементы строки разделяются пробелом или запятой, а строки отделяются точкой с запятой или записываются в разных строках матрицы (вектора). Так, неравенства

$$2x_1^3 - x_2 \le 5,$$
$$x_1 + x_2^2 \le 12$$

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 optimset) записывается параметр 'Algorithm' со значением 'interior-point' либо 'sqp', либо 'active-set'. По умолчанию применяется алгоритм '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 fminunc 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.MaxIter) или вычислений функции (options.FunEvals),

–1 – алгоритм завершен функцией output,

–2 – не найдено допустимой точки;

б) для алгоритмов доверительных областей и внутренней точки

2 – 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

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

г) только для алгоритма активного набора

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

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

д) только для алгоритма внутренней точки

-3 – текущая точка X ниже величины options.ObjectiveLimit и 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 и внутренней точки), firstorderopt, message. Пояснения требуют только новые поля: lssteplength – размер шага линейного поиска относительно направления поиска, constrviolation – максимальное значение 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.

Найти минимум функции

$$f(X) = (2 - x_1)^4 + (2x_1 - x_2)^2$$

при условиях

$$3x_1 + x_2 \le 8,$$
$$-x_1 + x_2 \le 3.$$

Сначала создадим *m*-файлы с именами primfun и primfun_g для вычисления целевой функции в процессе минимизации и построения контуров:

```
function f = primfun(x)
f=(2-x(1))^4+(2*x(1)-x(2))^2;
end
```

```
function f = primfun_g(x,y)
f=(2-x).^4+(2*x-y).^2;
end
```

Выпишем входные аргументы линейных ограничений:

$$A=\begin{bmatrix} 3 & 1 \\ -1 & 1 \end{bmatrix}, \quad b=\begin{bmatrix} 8 \\ 3 \end{bmatrix}.$$

Затем составим программу нахождения минимума и прорисовки линий уровня, границ допустимой области и траектории поиска в виде функции `runfmincon0`. В качестве алгоритма укажем `'interior-point'`, а для получения точек траектории напишем функцию `outfun` (укажем ее как значение параметра `'outputfcn'`).

```
function history = runfmincon0(x0)
 % Set up shared variables with OUTFUN
 history.x = [];
 history.fval = [];
 % call optimization
A=[3 1;-1 1];b=[8;3];

[X,Y]=meshgrid(0:0.02:3.5,0:0.02:6);Z=primfun_g(X,Y);
[c,h]=contour(X,Y,Z,[0.007 0.025 0.1 0.5 2 5 10 15 ...
20 27 35 45],'blue');
clabel(c,h,[0.5 5 15 27],'FontSize',7);
hold on;plot([8/3 1.25],[0 4.25],'g-','linewidth',2.3);
plot([0 1.25],[3 4.25],'g-','linewidth',2);
text(1,1.8,'The feasible region');
xlabel('x1'); ylabel('x2'); text(x0(1),x0(2)+0.2,'x0');
title('Minimization by fmincon');
options = optimset('outputfcn',@outfun,'display',...
'off','Algorithm','interior-point');
    [x,fval,exitflag,output]=fmincon(@primfun,x0,A,b,...
    [],[],[],[],[],options)
    y=history.x(:,1);z=history.x(:,2);
    hold on;plot(y,z,'b.',y,z,'r-');
```

```
function stop = outfun(x,optimValues,state)
     stop = false;
      switch state
         case 'init'
             hold on
         case 'iter'
         %  Concatenate  current  point  and  objective
         % function
           history.fval = [history.fval;...
           optimValues.fval];
           history.x = [history.x; x];
     end
  end
end
```

Вводим в командную строку

>> x0=[3 2];history=runfmincon0(x0)

и получаем результаты в численном и графическом виде (рис. 6):

```
x =
   1.6049    3.1852
fval =
   0.0250
exitflag =
    1
output =
        iterations: 13
        funcCount: 43
        constrviolation: 0
        stepsize: 5.6982e-007
        algorithm: 'interior-point'
        firstorderopt: 4.0000e-007
        cgiterations: 0
        message: [1x782 char]
```

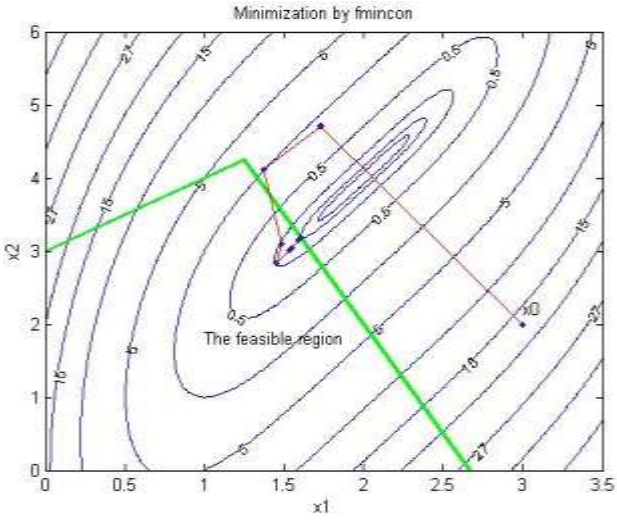


Рис. 6. Поиск минимума алгоритмом внутренней точки

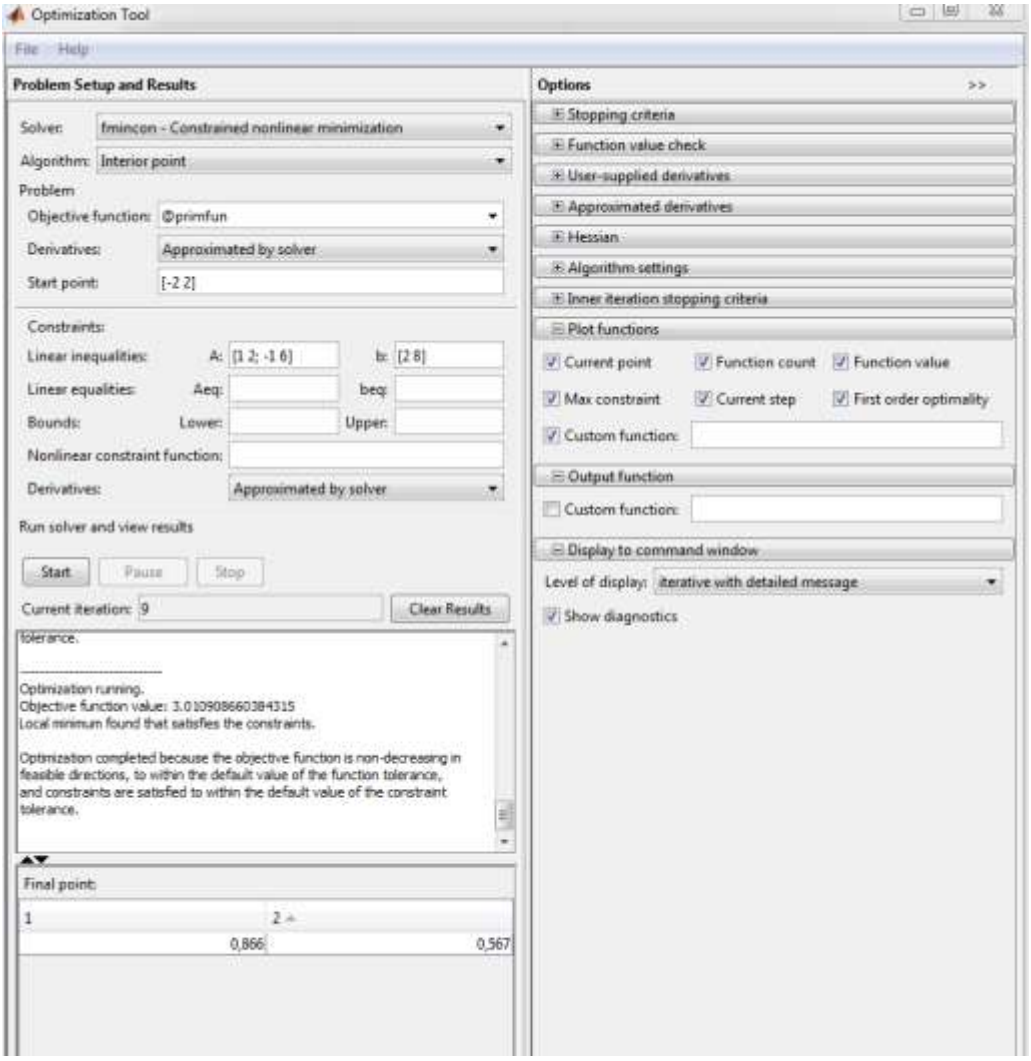
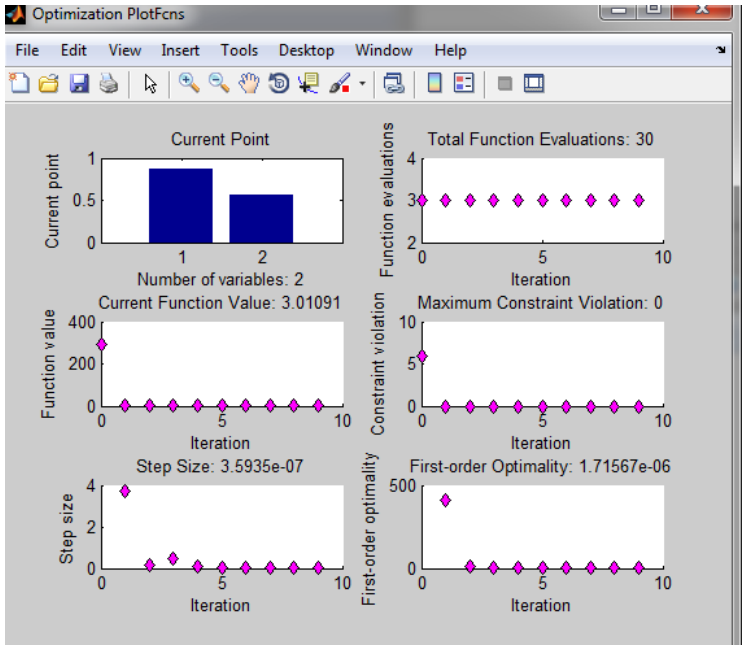


Рис 7. Моделирование в Optimization Tool




```
   End diagnostic information

                                                 First-order       Norm of
Iter F-count            f(x)  Feasibility        optimality           step
   0       3    2.920000e+02    6.000e+00         4.264e+02
   1       6    5.413739e+00    0.000e+00         4.098e+02      3.702e+00
   2       9    4.403052e+00    0.000e+00         7.875e+00      1.252e-01
   3      12    3.162356e+00    0.000e+00         2.032e+00      4.549e-01
   4      15    3.124962e+00    0.000e+00         3.718e-01      1.132e-01
   5      18    3.114113e+00    0.000e+00         8.752e-02      1.545e-02
   6      21    3.029552e+00    0.000e+00         1.596e-02      3.398e-02
   7      24    3.011092e+00    0.000e+00         2.814e-04      7.576e-03
   8      27    3.010909e+00    0.000e+00         8.275e-06      7.287e-05
   9      30    3.010909e+00    0.000e+00         1.716e-06      3.594e-07
```

Рис. 8. Результаты моделирования

Эти данные и рис. 6 показывают, что с заданной точностью достигнут локальный минимум на границе допустимой области в точке x_1 = 1,6049, x_2 = 3,1852 и с оптимальным значением целевой функции 0,025.

Заменив в `optimset` значение параметра `Algorithm` с `interior-point` на `sqp`, повторим поиск из той же начальной точки. Данные, представленные ниже, и рис. 7 показывают, что алгоритмом `SQP` минимум находится значительно быстрее.

```
>> x0=[3 2];history=runfmincon0(x0)
x =
   1.6049  3.1852
fval =
   0.0250
exitflag =
    1
output =
        iterations: 5
        funcCount: 21
        algorithm: 'sequential quadratic programming'
          message: [1x782 char]
          constrviolation: 0
          stepsize: 1
          firstorderopt: 5.6766e-007
```

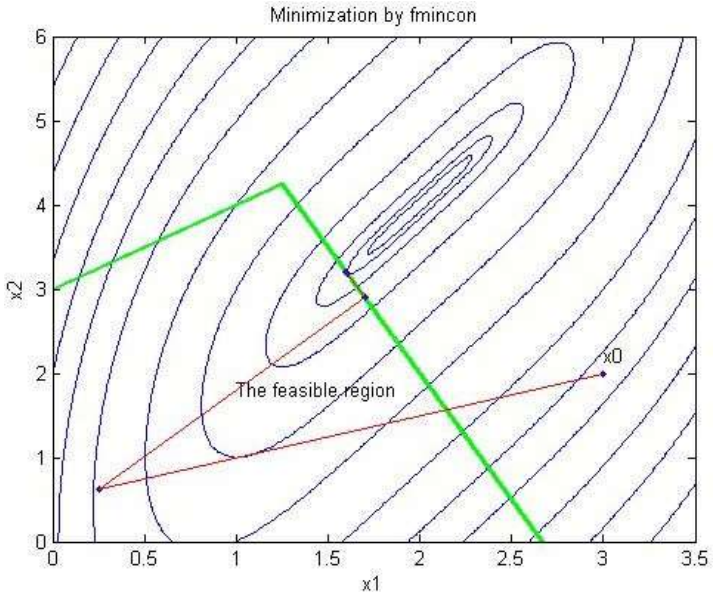


Рис. 7. Поиск минимума методом последовательного квадратичного программирования

Для точного вычисления градиента, используемого при минимизации, он задается аналитически в аргументе `fun`. Так, в рассмотренном выше примере функция `primfun` должна быть дополнена выражением градиента `g`:

```
function [f,g] = primfun(x)
```

```
    f=(2-x(1))^4+(2*x(1)-x(2))^2;
    g=[-4*(2-x(1))^3+4*(2*x(1)-x(2));-2*(2*x(1)-x(2))];
end
```

и в опции добавлен параметр `'Gradobj'` со значением `'on'`.

В следующем примере рассмотрим задачу с нелинейными ограничениями.

Пример 2.

Найти минимум функции

$$f(X) = (2 - x_1)^4 + (2x_1 - x_2)^2$$

при условиях

$$1{,}25(x_1 - 3{,}5)^2 + x_2 \le 5,$$

$$0{,}75 / x_1 - 1{,}5\sqrt{2x_1} + x_2 \le 0.$$

Нелинейные ограничения представим в *m*-файле с именем `primcon`:

```
function [c,ceq] = primcon(x)
   %Primer for fmincon
   c=[1.25*(x(1)-3.5)^2+x(2)-5;0.75/x(1)+x(2)-...
1.5*sqrt(2*x(1))];
    ceq=[];
end
```

Снова составим программу минимизации и прорисовки линий уровня, границ допустимой области и траектории поиска в виде функции `runfmincon00`.

```
function history = runfmincon00(x0)
history.x = [];
history.fval = [];
[X,Y]=meshgrid(0:0.02:3.5,0:0.02:6);Z=primfun_g(X,Y);
contour(X,Y,Z,[ 0.007 0.025 0.1 0.5 2 5 10 15 20 27  ...
35 45],'blue');
x=1.5:0.01:2.17; y=-1.25*(x-3.5).^2+5; hold on;
plot(x,y,'g-','linewidth',2.3); x=2.17:0.01:3.5;
y=1.5*sqrt(2*x)-0.75./x;hold on;plot(x,y,...
'g-','linewidth',2);
text(2.5,2,'The feasible region');
xlabel('x1'); ylabel('x2');
title('Minimization by fmincon');
options = optimset('outputfcn',@outfun,'display',...
'off', 'Algorithm', 'sqp');
[x,fval,exitflag,output]=fmincon(@primfun,x0,[],[],...
[],[],[],[],@primcon,options)
```

```
    y=history.x(:,1);z=history.x(:,2);
    hold on;plot(y,z,'b.',y,z,'r-');
function stop = outfun(x,optimValues,state)
     stop = false;
       switch state
         case 'init'
             hold on
         case 'iter'
           history.fval = [history.fval;...
           optimValues.fval];
           history.x = [history.x; x];
     end
 end
end
```

Здесь следует обратить внимание на входные аргументы функции `fmincon`.

После ввода и выполнения команды

>> x0=[0.5 4];history = runfmincon00(x0)

имеем нижеприведенные результаты и график траектории (рис. 8):

```
x =
   2.1661    2.7758
fval =
   2.4229
exitflag =
    1
output =
        iterations: 5
        funcCount: 24
        algorithm: 'sequential quadratic programming'
        message: [1x782 char]
        constrviolation: 1.9114e-012
        stepsize: 1
        firstorderopt: 6.5533e-008
```

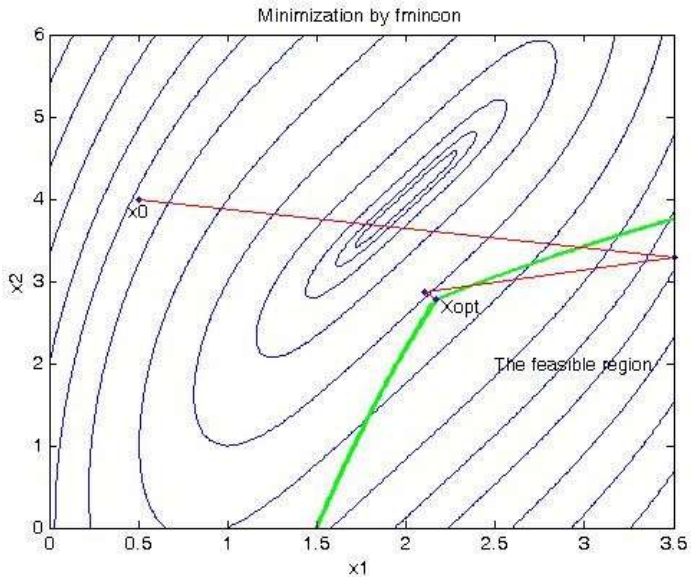


Рис. 8. Поиск условного минимума алгоритмом `SQP`
при нелинейных ограничениях

Как видно из рис. 8, оптимальное решение достигается снова на границе допустимой области.

В случае нелинейных ограничений помимо градиента целевой функции могут быть заданы градиенты ограничений в *m*-файле ограничений, и тогда для их использования необходимо добавить в опции параметр `'Gradconstr'` со значением `'on'`.

В приведенных примерах с `fmincon` решались задачи с одноэкстремальными целевыми функциями. Теперь рассмотрим пример поиска минимума многоэкстремальной функцией при двухсторонних ограничениях на переменные.

Пример 3.

Найти минимум функции

$$f = 3(1-x_1)^2\exp(-x_1^2-(x_2+1)^2) - 10(x_1/5 - x_1^3 - x_2^5)\exp(-x_1^2 - x_2^2) - 1/3\exp(-(x_1+1)^2 - x_2^2)$$

на параллелепипеде

$$-3 \le x_j \le 3,\ j = 1, 2.$$

Для решения этой задачи сначала запишем в *m*-файл целевую функцию

```
function f=myfun(x)
f=3*(1-x(1))^2*exp(-x(1)^2-(x(2)+1)^2)-10*(x(1)/5....
-x(1)^3-x(2)^5)*exp(-x(1)^2-x(2)^2)-1/3*exp(-(x(1)...
+1)^2-x(2)^2);
end
```

Затем составляем программу поиска минимума и отображения целевой функции, ее линий уровня и траектории поиска в виде функции от начальной точки:

```
function history = runfmincon5_1(x0)
history.x = [];
history.fval = [];
% call optimization
options = optimset...
('outputfcn',@outfun,'display','off','Algorithm',. ..'in-
terior-point');
[x,fval,exitflag,output]= fmincon(@myfun,x0,[],[],[],...
[],[-3 -3],[3 3],[],options)
 [X,Y]=meshgrid(-3:0.02:3);Z=3*(1-X).^2.*exp(-X.^2- ...
(Y+1).^2)-10*(X/5 ...
-X.^3-Y.^5).*exp(-X.^2-Y.^2)-1/3*exp(-(X+1).^2-Y.^2);
meshc(X,Y,Z);
    y=history.x(:,1);z=history.x(:,2);v=history.fval;
    hold on;plot3(y,z,v,'black.',y,z,v,'red-');
    v=-10*ones(size(z));hold on;plot3(y,z,v,'red-');
    xlabel('x1'); ylabel('x2');zlabel('f');
    title('Sequence of Points Computed by fmincon');

 function stop = outfun(x,optimValues,state)
     stop = false;
      switch state
         case 'init'
             hold off
         case 'iter'
  % Concatenate current point and objective function
         history.fval = [history.fval; optimValues.fval];
         history.x = [history.x; x];
     end
 end
end
```

После ввода и выполнения команды

```
>> x0=[1 1]; history = runfmincon5_1(x0)
```

получаем

```
x =
    0.2283  -1.6255
fval =
   -6.5511
exitflag =
     1
output =
        iterations: 16
        funcCount: 56
        constrviolation: 0
        stepsize: 4.8893e-008
        algorithm: 'interior-point'
        firstorderopt: 1.2023e-007
        cgiterations: 0
        message: [1x782 char]
```

На рис. 9 показан процесс поиска минимума. Как видно из представленных результатов, он оказался успешным.

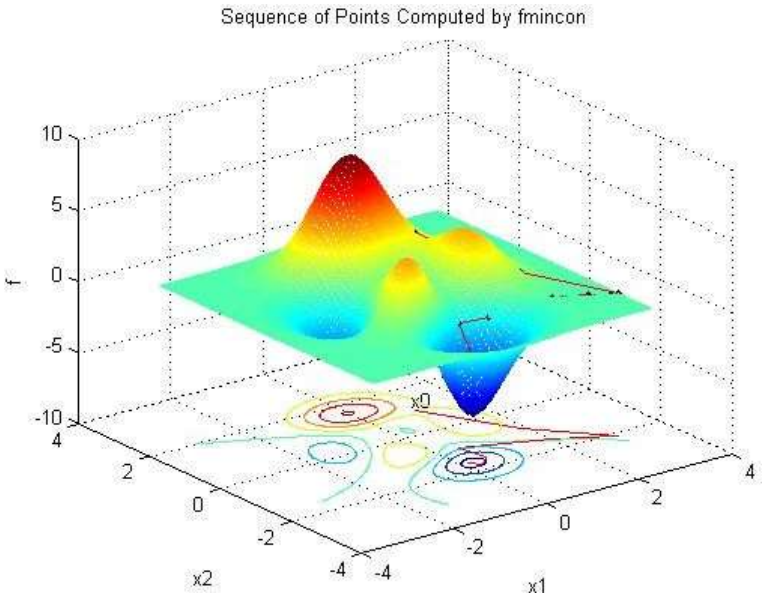


Рис. 9. Вид целевой функции и траектория движения к минимуму

Однако этот результат следует считать случайным, так как случайно выбрана данная начальная точка.

Повторив поиск из другой начальной точки, получаем

```
>> x0=[-0.5 3]; history = runfmincon5_1(x0)
x =
  -2.9984    2.9976
fval =
 3.2869e-005
exitflag =
     1
output =
        iterations: 27
        funcCount: 84
         constrviolation: 0
         stepsize: 7.2753e-004
         algorithm: 'interior-point'
         firstorderopt: 3.2032e-007
         cgiterations: 0
         message: [1x782 char]
```

Из показателей поиска (признака `exitflag`, критериев точности) как будто следует, что получено решение задачи. На самом деле поиск завершился даже не в точке одного из локальных минимумов, что хорошо видно на рис. 10.

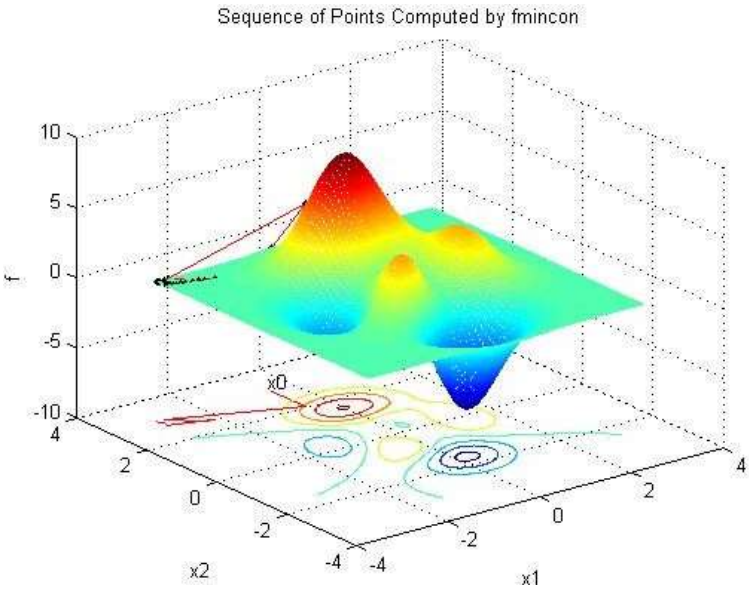


Рис. 10. Поиск минимума из точки (–0,5 3)

Примечание. В Toolbox Optimization имеется функция rtrlink, по функциональности аналогичная fmincon. Она применима для решения задач без ограничений и с линейными и нелинейными ограничениями. В ней также используются алгоритмы active-set и interior-point. Однако она требует установки самостоятельной библиотеки KNITRO, к которой происходит обращение при вызове rtrlink.

Повышение надежности отыскания минимума многоэкстремальной функции требует применения методов глобального поиска. Некоторые из них, включенные в MATLAB, рассматриваются в следующем разделе.

Задания к лабораторной работе

1. Ознакомиться с вариантами обращения к функции fmincon, ее входными и выходными аргументами.
2. Воспроизвести пример 1, исследовать работу трех алгоритмов из разных начальных точек, принадлежащих и не принадлежащих допустимой области.
3. То же в условиях примера 2. Кроме того, повторить нахождение минимума для одной начальной точки без использования графики.
4. Найти минимум функции из примера 2 при условиях, включающих ограничения из примеров 1 и 2.
5. Воспроизвести пример 3 и исследовать работу всех алгоритмов fmincon из нескольких начальных точек. Найти максимум этой же целевой функции, используя разные алгоритмы.
6. **Провести анализ результатов работы алгоритмов**.
7. **Решить задачу условной оптимизации по вариантам. Использовать функцию** fmincon. Исследовать работу алгоритмов ***Trust Region, Active Set,*** SQP, ***Interior-point.***

Варианты к работе:

Решить задачи по вариантам.

1. $z = \sqrt{x_1^2 + x_2^2}$

при условиях:

$$\begin{cases} 3x_1 + 4x_2 \le 24 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

2. $z = (x_1 - 2)^2 + (x_2 - 3)^2$

при условиях:

$$\begin{cases} x_1 + 2x_2 \le 12 \\ x_1 + x_2 \le 9 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

3. $z = (x_1 - 4)^2 + (x_2 - 6)^2$

при условиях:

$$\begin{cases} 2x_1 + 3x_2 \le 12 \\ x_1 + x_2 \ge 1 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

4. $z = x_1 \cdot x_2$

при условиях:

$$\begin{cases} 2x_1 + x_2 \le 10 \\ x_1 + x_2 \le 6 \\ x_1 + 2x_2 \ge 2 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

5. $z = 2x_1 + 3x_2 - 2x_1^2$

при условиях:

$$\begin{cases} x_1 + 2x_2 \le 4 \\ x_1 + x_2 \le 2 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

6. $z = (x_1 - 6)^2 + (x_2 - 2)^2$

при условиях:

$$\begin{cases} x_1 + 2x_2 \le 8 \\ 3x_1 + x_2 \le 15 \\ x_1 + x_2 \ge 1 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

7. $z = x_1 + 3x_2$

при условиях:

$$\begin{cases} (x_1 - 5)^2 + (x_2 - 3)^2 \ge 9 \\ (x_1 - 5)^2 + (x_2 - 3)^2 \le 36 \\ x_1 + x_2 \ge 8 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

8. $z = 2(x_1 - 5)^2 + (x_2 - 7)^2$

при условиях:

$$\begin{cases} x_1 + x_2 \le 9 \\ x_1 + 2x_2 \le 19 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

9. $z = 2x_1 + x_2 - x_1^2$
при условиях:

$$\begin{cases} 3x_1 + 2x_2 \le 12 \\ x_1 \ge 0 \\ 0 \le x_2 \le 3 \end{cases}$$

10. $z = 2(x_1 - 7)^2 + 4(x_2 - 3)^2$
при условиях:

$$\begin{cases} x_1 + 2x_2 \ge 2 \\ x_1 + x_2 \le 6 \\ 2x_1 + x_2 \le 10 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

11. $z = x_1 + x_2$
при условиях:

$$\begin{cases} (x_1 - 5)^2 + (x_2 - 3)^2 \ge 9 \\ (x_1 - 5)^2 + (x_2 - 3)^2 \le 36 \\ x_1 + x_2 \ge 8 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

12. $z = x_1 \cdot x_2$
при условиях:

$$\begin{cases} x_1 + 2x_2 \ge 2 \\ x_1 + x_2 \le 6 \\ 2x_1 + x_2 \le 10 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

13. $z = 2x_1 + 3x_2 - 0{,}2x_1^2 - 0{,}2x_2^2$
при условиях:

$$\begin{cases} x_1 + 3x_2 \le 13 \\ 2x_1 + x_2 \le 10 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

14. $z = x_1 + x_2$
при условиях:

$$\begin{cases} (x_1 - 5)^2 + (x_2 - 3)^2 \ge 9 \\ (x_1 - 5)^2 + (x_2 - 3)^2 \le 36 \\ x_1 + x_2 \ge 8 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

15. $z = 3x_1 + 6x_2 - 0{,}3x_1^2 - 0{,}3x_2^2$
при условиях:

$$\begin{cases} 9x_1 + 8x_2 \le 72 \\ x_1 + 2x_2 \le 10 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$

16. $z = 3x + y$
при условиях:

$$\begin{cases} x^2 + y^2 \le 40 \\ x^2 + y^2 \ge 4 \\ x_1 \ge 0 \\ x_2 \ge 0 \end{cases}$$