**Лабораторная работа № 4-5**

**Оптимизация**

**Нелинейные задачи**

Нелинейное программирование не исчерпывает класс нелинейных задач, которые могут быть решены в Optimization Toolbox. Функции Toolbox fgoalattain, fminmax, fseminf позволяют исследовать задачи о достижении границы, находить решение в задаче о минимаксе. Ниже приведены формулировки задач и интерфейс функций, предназначенных для их решения. Область допустимых значений независимых переменных может задаваться как линейными, так и нелинейными ограничениями, в общем случае они совпадают с ограничениями в задаче нелинейного программирования, рассмотренной ранее.

**Задача о достижении границы**

В задаче о достижении границы задана вектор-функция *F*(*x*) и два вектора *w* и *g*. Требуется найти х из области допустимых значений, для которого величина у будет минимальной при выполнении условий *Fi*(*x*) – *wi*γ < *gi* для всех компонент *F*(*x*), *w* и *g*.

Поставленная задача решается при помощи функции fgoalattain, обращение к которой в достаточно общем случае имеет вид:

х = fgoalattain (fun, х0, g, w, A, b, Aeq, beq, lb, ub, nonlcon, options, P1, P2, ...)

Входной аргумент fun является указателем на файл-функцию (или ее именем), вычисляющей вектор-функцию F(x). Аргументы g и w – векторы, длина которых совпадает с числом значений вектор-функции. Использование остальных аргументов аналогично fmincon. Интерфейс функции fgoalattain, так же как и других функций Optimization Toolbox, допускает вызов с переменным числом входных и выходных аргументов в зависимости от условий, определяющих область допустимых значений, и требуемого результата. Так, например, в случае нелинейных ограничений и необходимости вычислить не только х, но F(x) и у, следует указать три выходных аргумента, а неиспользуемые входные сделать пустыми массивами:

[х, F, gamma] = fgoalattain (fun, x0, g, w, [ ], [ ], [ ], [ ], [ ], [ ], nonlcon)

Четвертый дополнительный выходной аргумент функции fgoalattain служит для получения информации о вычислительном процессе. Положительное его значение свидетельствует об успешном завершении вычислений, ноль – вычисления остановлены из-за достижения максимального числа итераций или превышения максимально возможного числа вызовов исследуемой функции, а отрицательное значение сигнализирует о том, что решение не найдено.

В справочной системе MATLAB по Optimization Toolbox приведен пример конструирования регулятора с обратной связью с привлечением функции

fgoalattain (см разд. Optimization Toolbox: Tutorial: Multiobjective Examples).

**Минимизация функции с полубесконечными ограничениями**

Постановка задачи заключается в поиске минимума функции нескольких переменных (16.10) в области, задаваемой в общем случае линейными ограничениями (16.2 – 16.4), нелинейными (16.11) и полубесконечными ограничениями вида

|  |  |  |
| --- | --- | --- |
|  |  | (16.12) |

где *wi* – векторы дополнительных переменных размерности не более двух, компоненты которых изменяются в заданном интервале.

Для решения этой задачи предназначена функция fseminf со следующим интерфейсом:

X = fseminf(fun, x0, ntheta, seminfcon, A, b, Aeq, beq, lb, ub, options,P1, P2, ...)

причем могут быть указаны дополнительные выходные аргументы. Во второй из них запишется значение функции в точке минимума, а в третий – информация о вычислительном процессе. Смысл значений третьего выходного аргумента тот же самый, что и для функции fmincon, рассмотренной выше.

Входные аргументы функции fseminf: fun, x0, А, b, Aeq, beq, lb, ub предназначены для указания минимизируемой функции, начального приближения, матриц и векторов, определяющих линейные ограничения типа неравенств и равенств (16.2–16.4). Обсудим назначение параметров, непосредственно связанных с нелинейными ограничениями (16.11) и полубесконечными (16.12). Количество ограничений вида (16.12) задается в ntheta, а все нелинейные ограничения, включая полубесконечные, программируются в файл-функции, указатель на которую (или ее имя) передается в seminfcon. Эта файл-функция должна иметь специальный интерфейс и структуру (листинг 16.5).

Листинг 16.5. Общий вид файл-функции для полубесконечных ограничений

function [c, ceq, K1, K2,..., Kntheta, S] = sem\_con(x, S)

% Проверка, был ли инициализирован массив шагов S

% для дополнительных переменных

if isnan(S(l, 1))

% Это первый вызов sem\_con, следует инициализировать массив S

S(l, 1) = ...; S(1, 2) = ...;

S(ntheta ,1) = ...; S(ntheta, 2) =...;

end

% Генерация сетки значений дополнительных переменных

% Если дополнительная переменная является вектором с двумя компонентами,

% то следует использовать meshgrid

wl =...;

wntheta = ...;

% Вычисление левых частей полубесконечных ограничений (16.12)

% в узлах сетки

K1 =...;

 ...

Kntheta =...;

% Вычисление левых частей нелинейных ограничений

% (равенств и неравенств) типа (16.11)

c = ...;

ceq = ...;

Для заданного *х* файл-функция sem\_con должна возвращать значения вектор-функций *с*(*x*) и *ceq*(*x*), входящих в левые части нелинейных ограничений (16.11). При отсутствии ограничений этого типа следует присвоить *с* и *ceq* пустые массивы в теле файл-функции. Список выходных аргументов sem\_con включает в себя также значения левых частей *Кi(х*, *wi*) полубесконечных ограничений (16.12), вычисленных для некоторого набора значений каждой из переменных wi. Эти значения выбираются на сетке с постоянным шагом, причем шаг для wi записан в *i-*ой строке массива s. Если wi – скаляр, то выходной аргумент, соответствующий *К*i(*х,* *wi*), будет вектором и шаг определяется значением *S*(*i*, l), а элемент *S*(*i*, 2) не используется. В случае, когда аргумент wi функции *Ki(x,* *wi*) является вектором длины 2, для wi генерируется сетка значений с шагами *S*(*i*, l) и *S(i*, 2) по каждой из компонент при помощи функции meshgrid.

В ходе вычислений функция fseminf обращается к sem\_con с рекомендуемыми значениями шагов, но при первом вызове semcon массив *S* должен быть инициализирован. Первый вызов sem\_con легко отличить от последующих, поскольку входной аргумент *S* содержит NaN (не числа). Поэтому в начале файл-функции sem\_con размещен условный оператор if, который позволяет инициализировать массив шагов.

Рассмотрим на простом примере минимизацию нелинейной функции с полубесконечными ограничениями. Пусть требуется найти минимум функции:



при отсутствии линейных и нелинейных ограничений, но с заданным полубесконечным ограничением вида:



Для решения поставленной задачи требуется написать файл-функции для вычисления f(x1, x2) и K1(х1, x2, w1). Целевая функция программируется просто (листинг 16.6). При создании файл-функции для полубесконечного ограничения следуйте приведенным выше соглашениям относительно ее интерфейса и алгоритма. В нашем примере всего одно полубесконечное ограничение, причем дополнительная независимая переменная *wi* является скаляром. В случае возникновения затруднений обратитесь к листингу 16.7.

Листинг 16.6. Минимизируемая функция решения задачи с полубесконечными i ограничениями

function f = myfuns(x)

% целевая функция

f = x(1)^2 + 2\*x(2)^2;

function [c, ceq, K1, S] = sem\_con(X, S)

if isnan(S(1, 1))

% Инициализация шага сетки 0.2 для скалярной переменной w1

% (вторая компонента не будет использоваться)

S = [0.2 0];

end

% Генерация сетки на отрезке [0, 100] для полубесконечных ограничений

w1 = 1:S(1, 1):100;

% Вычисление полубесконечного ограничения в узлах сетки

K1 = cos(w1\*X(1))./(w1\*X(2)) - 0.1;

% Нелинейные ограничения отсутствуют, задаем пустой массив

c= [];

ceq = [];

Для решения задачи осталось задать начальное приближение

>> x0= [0.5; 0.2];

и вызвать функцию fseminf, установив число полубесконечных ограничений в 1:

>> [x, fval] = fseminf(@myfuns, x0, 1, @sem\_con)

Optimization terminated: magnitude of search direction less than 2\*options.TolX

and maximum constraint violation is less than options.TolCon.

Active inequalities (to within options.TolCon = 1e-006):

 lower upper ineqlin ineqnonlin

 1

 2

x =

 1.3512

 2.1785

fval =

 11.3175

>>

**Минимаксная задача**

Постановка минимаксной задачи заключается в нахождении



для нелинейной вектор-функции *f*(x) = (f1(x), f2(x), …, fn(x)) при ограничениях (16.2–16.4) и (16.11) тех же самых, что и в задаче нелинейного программирования. Для решения данной задачи предназначена функция fminimax, применение которой в достаточно общем случае выглядит следующим образом:

х = fminimax(fun, х0, А, b, Aeq, beq, lb, ub, nonlcon, options, P1, P2, ...)

Первый входной аргумент fminimax является файл-функцией, возвращающей вектор значений {*Fi*(*x*)}. Смысл остальных входных аргументов такой же, как и для fmincon. Интерфейс функции fminimax, как и большинства функций Optimization Toolbox, допускает переменное число входных и выходных аргументов, причем в качестве неиспользуемых входных аргументов указывается пустой массив. Во втором дополнительном выходном аргументе fminimax возвращаются значения вектор-функции *f*(*х*), в третьем – максимальное из них, а четвертый выходной аргумент содержит информацию о причине останова вычислений. Смысл его значений тот же, что и для функции fmincon.

Рассмотрим модельную задачу для демонстрации обращения к функции fminimax, в которой надо найти независимые переменные *х*1 и *х*2 из условия:



где



при линейных ограничениях



и нелинейном ограничении



Для решения задачи напишите файл-функцию для вычисления минимизируемой вектор-функции, файл-функцию для задания нелинейных ограничений и файл-программу, реализующую обращение к fminimax. В листингах 16.8–16.10 приведены тексты программ.

Листинг 16.8. Минимизируемая функция решения задачи о минимаксе

function f = vectfun3(x)

% Вычисление минимизируемой вектор-функции

f(1) = (x(1) - 1)^2 + (x(2) - 2)^2 - 2.1;

f(2) = x(1)^1.3 + 2\*x(2)^0.9 - 1;

f (3) = 1.7\*x(1) + x(2) - 0.25;

Листинг16:9: Функция вычисления ограничений в задаче о минимаксе

function [c, ceq] = nonlinear(x)

% Задание нелинейных ограничений

% ограничений-неравенств нет

c= [ ];

% ограничения-равенства

ceq(1) = -exp(x(1)) + exp(x(2)) - 0.5;

Листинг16.10: m-файл test\_mm решения задачи о минимаксе

% начальное приближение

x0 = [1; 0];

% задание ограничений-равенств

A = [3 2];

b = 2.2;

% Задание ограничений снизу на переменные

lb = [0; 0];

[x, fval]=fminimax(@vectfun3,x0,A,b,[],[],lb,[],@nonlinear)

При вызове m-файла test\_mm получается результат:

Optimization terminated: magnitude of search direction less than 2\*options.TolX

 and maximum constraint violation is less than options.TolCon.

Active inequalities (to within options.TolCon = 1e-006):

 lower upper ineqlin ineqnonlin

 1

 3

x =

 0.2048

 0.5466

fval =

 0.6448 0.2885 0.6448

>>

**Решение нелинейных уравнений**

Optimization Toolbox позволяет решить системы нелинейных уравнений, общий вид которых задается при помощи нелинейной вектор-функции: *F*(*х*) = 0, а покомпонентная запись имеет вид



Первым аргументом fsolve является указатель на файл-функцию (или ее имя), вычисляющую F(x). Данная файл-функция принимает на входе вектор аргументов и возвращает вектор значений F(x). Начальное приближение указывается во втором аргументе fsolve.

В достаточно общем случае вызов fsolve выглядит следующим образом:

х = fsolve(fun, x0, options, P1, P2, ...)

где options – управляющая структура, а необязательные аргументы P1, P2,... есть значения параметров, от которых может зависеть левая часть системы уравнений. Если в списке входных аргументов fsolve всего два входных аргумента, то в командное окно выводится предупреждение, связанное с тем, что в версии 2.0 Optimization Toolbox алгоритм fsolve был модифицирован.

Для подавления этого предупреждения следует вызывать fsolve с управляющей структурой, которую можно сформировать, например, со всеми принятыми по умолчанию настройками при помощи optimset:

х = fsolve (fun, x0, optimset ('fsolve'))

Функция fsolve может быть вызвана с несколькими выходными аргументами. Второй выходной аргумент содержит значения вектор-функции для найденного решения, а третий – сведения о работе алгоритма. Отрицательное значение третьего аргумента говорит о том, что решение не найдено из-за расходимости вычислительного процесса. Ноль означает досрочное прерывание вычислений при достижении максимально допустимого числа итераций или обращений к вектор-функции системы. Нахождение решения с заданной точностью подтверждается положительным значением третьего выходного аргумента fsolve. Решите систему из двух нелинейных уравнений с двумя неизвестными



Файл-функция, соответствующая левой части системы, программируется просто (листинг 16.11).

Листинг 16.11. Файл-функция, вычисляющая левую часть системы уравнений (6.13)

function F = mysys(x)

F(1) = x(1)\*(2 - x(2)) - cos(x(1))\*exp(x(2));

F(2) = 2 + x(1) - x(2) - cos(x(1)) - exp(x(2));

Указание начальной точки (0, 0) в функции fsoive приводит к следующему

результату:

>> [x, f] = fsolve(@mysys, [0 0], optimset('fsolve'))

Optimization terminated: first-order optimality is less than options.TolFun.

x =

 0.7391 0.4429

f =

 1.0e-011 \*

 -0.4702 -0.6404

>>

Мы рассмотрели пример, в котором число неизвестных совпадает с числом уравнений системы. Алгоритмы, заложенные в функцию fsolve, допускают несовпадение числа неизвестных и уравнений. В качестве упражнения решите системы:



При исследовании первой системы задавайте различные начальные приближения и убедитесь, что возможно найти приближенное решение, достаточно близкое к точному *х*1 = *х*2 =1. Для второй системы выбор начального приближения определяет, для какого точного решения разыскивается приближенное – сравните, например, начальные приближения [1 1 1] и [0 1 1].

Алгоритмы, реализованные в функции fsolve, основаны на минимизации некоторых критериев, связанных с исходной системой нелинейных уравнений, в частности, суммы квадратов компонент вектор-функции. Решением считается та точка, в которой значение критерия мало, что не гарантирует существования корней системы в окрестности найденной точки. В этом легко убедиться на простом примере – примените функцию fsolve для системы из одного уравнения *х*2 + 10-6 = 0, не имеющего вещественных корней. Используйте для сравнения fzero с некоторым начальным приближением к корню. Функция fzero пытается отделить интервал, на границах которого непрерывная функция *х*2 + 10-6 имеет разные знаки для гарантии существования корня. Разумеется, такого интервала не существует и работа fzero прерывается.

Одним из алгоритмов функции fsolve является метод наименьших квадратов, который также используется для решения ряда оптимизационных задач, в частности, подбора параметров.

**Метод наименьших квадратов**

Метод наименьших квадратов (МНК) применяется, например, для решения систем линейных уравнений, в которых число неизвестных не совпадает с числом уравнений.

Функция lsqnonneg предназначена для поиска только неотрицательных решений систем *Сх* = *d*, т. е. векторов *х*, все компоненты которых больше либо равны нулю. По существу, при помощи метода наименьших квадратов решается задача на нахождение минимума  среди всех *xi* > 0.

Через || || обозначена вторая векторная норма, являющаяся квадратным корнем из суммы квадратов компонентов вектора. Достаточно общий вариант вызова lsqnonneg выглядит следующим образом:

[х, resnorm, residual, flag] = lsqnonneg(С, d, x0)

где первые два входных аргумента содержат матрицу и вектор системы, хо – начальное приближение, в х возвращается решение, а в дополнительных выходных аргументах возвращаются норма невязки (resnorm), вектор невязки (flag) и информация о завершении процесса вычислений (flag). Положительное значение переменной flag подтверждает сходимость итерационного процесса к решению, а нулевое значение предупреждает о превышении максимально допустимого числа итераций или вычислений квадрата нормы невязки, установленных по умолчанию.

Выходные аргументы и х0 являются необязательными параметрами, если значение х0 не указано, то по умолчанию используется нулевое начальное приближение.

Функция lsqlin позволяет найти решение более общей задачи



с линейными ограничениями на решение (16.2–16.4).

**Подбор параметров**

Предположим, что в некоторый физический закон у = F(a1, a2, а3, а4, х) входят неизвестные параметры а1, а2, а3 и а4. Проделан ряд экспериментов и получено п опытных данных (хйсй{, ydatt) с целью установления значений параметров. Возникает вопрос, как выбрать параметры физического закона так, чтобы результаты эксперимента соответствовали ему некоторым наилучшим образом.

Решение задачи о подборе параметров в Optimization Toolbox основано на методе наименьших квадратов, который в данном случае состоит в нахождении минимума выражения



по всевозможным значениям а1, а2, а3 и а4. Функция lsqcurvefit предназначена для решения задачи о подборе параметров (число параметров может быть произвольным). Обращение к ней практически не отличается от вызова других функций Toolbox и в самом простом случае имеет вид:

х = lsqcurvefit(fun, a0, xdat, ydat)

где fun – указатель на файл-функцию (или ее имя), которая вычисляет F(a1, a2, а3, а4, х). Список ее входных аргументов должен содержать массив для передачи значений параметров в файл-функцию и независимую переменную х.

Дополнительно могут быть поставлены ограничения на параметры, которые в векторной записи имеют вид lb < а < ub. В этом случае векторы lb и ub задаются в пятом и шестом аргументах lsqcurvefit:

х = lsqcurvefit(fun, a0, xdat, ydat, lb, ub)

Определите параметры в следующем примере. Пусть

a в результате эксперимента получены следующие значения:

xdat = [0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0];

ydat = [1.1 2.1 3.5 3.9 4.3 5.1 4.2 4.0 3.3 2.2 2.1].

Известно, что значение каждого из параметров может находиться в промежутке [-10, 10]. Первым шагом является создание файл-функции (листинг 16.12). Параметры в fitfun передаются через вектор а.

Листинг 16.12. Файл-функция, зависящая от вектора параметров и аргумента

function y = fitfun(a, x)

y = a(1)\*exp(a(2)\*x) + a(3)\*sin(a(4)\*x);

Теперь следует задать векторы xdat и ydat, отобразить данные на графике, выбрать начальное приближение а0, построить график при начальном приближении, задать границы и вызвать lsqcurvefit. После отыскания параметров необходимо вывести график функции и убедиться, что функция с найденными параметрами достаточно точно описывает данные. Последовательность команд, приведенная в листинге 16.13, реализует вышеописанные действия, в результате получается график, изображенный на рис. 16.1. В командное окно выводится вектор найденных значений параметров:

а =

1.0959 1.1678 2.7003 3.7505

Листинг 16.13: Подбор параметра

% Ввод данных

xdat =0:0.1:1;

ydat = [1.1 2.1 3.5 3.9 4.3 4.6 4.2 4.0 3.3 2.2 2.1];

% Отображение данных на графике

plot(xdat,ydat,'o');

grid on

% Выбор начального приближения

a0 = [0.0 0.0 4.0 1.0];

% Построение графика функции от начального приближения

x = [0:0.05:1];

ya0 = fitfun(a0, x);

hold on;

plot(x,ya0,'--b')

% Задание границ области параметров

LB = [-10 -10 -10 -10];

UB = [10 10 10 10] ;

% Подбор параметров, точка с запятой в конце команды не ставится для

% вывода результата в командное окно

a = lsqcurvefit(@fitfun, a0, xdat, ydat, LB, UB)

% Визуализация функции с найденными значениями параметров

ya = fitfun(a, x) ;

Hfit = plot(x, ya);

set(Hfit, 'LineWidth', 2)

legend('данные', 'начальное приближение', 'результат', 4)



Рис. 16.1. Результат подбора параметров

Обратите внимание, что начальное приближение оказывает существенное влияние на получаемый результат. Если, к примеру, в качестве начального приближения к искомым параметрам взять вектор а0 = [4.0 -1.0 0.0 0.0], то подбор параметров не приведет к хорошему результату (рис. 16.2).



Рис. 16.2. Результат подбора параметров

**Параметры оптимизации**

Задачи оптимизации условно разделены в Optimization Toolbox на два класса: Medium-scale (средние) и Large-scale (большие), в зависимости от размерности задачи, т. е. числа переменных. Для решения каждого из классов задач реализованы соответствующие численные методы, объяснение которых выходит за рамки данной книги. Впрочем, краткая информация об алгоритмах содержится в справочной системе MATLAB по Toolbox. Пользователь имеет возможность отслеживать ход вычислительного процесса и задавать параметры, управляющие вычислениями.

Для установки параметров следует сформировать структуру options при помощи функции optimset и затем указать данную структуру в качестве входного аргумента функции Optimization Toolbox, выбранной для решения поставленной задачи (формирование структуры options для функций fzero, fminbnd и fminsearch описано в разд. "Задание дополнительных параметров").

Функции, предназначенные для решения задач оптимизации, обладают достаточно большим набором параметров. Команда optimset, вызванная с именем функции в качестве входного аргумента, выводит в командное окно структуру с информацией о текущих опциях вычислительного алгоритма данной функции. Обращение к optimset без входных аргументов позволяет отобразить в командном окне все возможные значения каждого из параметров. Подробная информация о назначении каждого параметра приведена в справочной системе MATLAB (см. разд. Optimization Toolbox: Function Reference: Optimization Parameters, где приведена таблица со всеми параметрами и описаны функции Toolbox).

Получите установки функции fsolve, использовав обращение optimset ('fsolve'). В командном окне отображаются параметры и их значения, ниже приведены параметры, используемые в данном разделе.

>> optimset ('fsolve')

ans =

 Display: 'final'

 MaxFunEvals: [1x21 char]

 MaxIter: 400

 TolFun: 1.0000e-006

 TolX: 1.0000e-006

 FunValCheck: 'off'

 OutputFcn: []

 PlotFcns: []

 ActiveConstrTol: []

 BranchStrategy: []

 DerivativeCheck: 'off'

 Diagnostics: 'off'

 DiffMaxChange: 0.1000

 DiffMinChange: 1.0000e-008

 GoalsExactAchieve: []

 GradConstr: []

 GradObj: []

 Hessian: []

 HessMult: []

 HessPattern: []

 HessUpdate: []

 InitialHessType: []

 InitialHessMatrix: []

 Jacobian: 'off'

 JacobMult: []

 JacobPattern: [1x25 char]

 LargeScale: 'off'

 LevenbergMarquardt: []

 LineSearchType: 'quadcubic'

 MaxNodes: []

 MaxPCGIter: [1x33 char]

 MaxRLPIter: []

 MaxSQPIter: []

 MaxTime: []

 MeritFunction: []

 MinAbsMax: []

 NodeDisplayInterval: []

 NodeSearchStrategy: []

 NonlEqnAlgorithm: 'dogleg'

 NoStopIfFlatInfeas: []

 PhaseOneTotalScaling: []

 Preconditioner: []

 PrecondBandWidth: 0

 RelLineSrchBnd: []

 RelLineSrchBndDuration: []

 ShowStatusWindow: []

 Simplex: []

 TolCon: []

 TolPCG: 0.1000

 TolRLPFun: []

 TolXInteger: []

 TypicalX: [1x25 char]

>>

Назначение параметров Display, MaxFunEvals, MaxIter, TolFun, TolX такое же, как и у функций fzero, fminbnd и fminsearch.

Следует иметь в виду, что все параметры делятся на три группы в зависимости от размерности задачи.

• Параметры установки Large-scale алгоритмов.

• Параметры установки Medium-scale алгоритмов.

• Общие параметры для Large- и Medium-scale алгоритмов.

Применяемый алгоритм зависит от значения LargeScale, 'on' разрешает использование Large-scale алгоритма, если он допустим для решаемой задачи. Имеются некоторые ограничения на область применения Large-scale алгоритмов, в частности, получающаяся в ходе решения система уравнений не должна быть недоопределенной, т. е. число уравнений не может превосходить число неизвестных.

Такие параметры, как JacobPattern, LevenbergMarquardt, MaxPCGIter, TolPCG, PrecondBandwidth, соответствуют Large-scale алгоритмам. Получающиеся в процессе вычислений системы линейных уравнений решаются итерационным методом – методом предобусловленных сопряженных градиентов (PCG). По умолчанию выбирается диагональный предобусловливатель, т. е. ширина верхней полуленты равна нулю. В ряде случаев возможно добиться ускорения сходимости итерационного процесса за счет увеличения ширины ленты предобусловливателя. Параметр PrecondBandwidth служит для задания ширины верхней полуленты ленты в исходной матрице, на основе которой строится предобусловливатель. Максимальное число итераций в методе сопряженных градиентов определяется значением MaxPCGIter, а критерий останова – TolPCG.

Задание матрицы Якоби вектор-функции исследуемой задачи значительно ускоряет вычисления. Файл-функция, вычисляющая левую часть исследуемой системы нелинейных уравнений (например, в случае fsolve), может иметь два выходных аргумента, во втором возвращается матрица Якоби системы. Соответствующий пример разобран ниже. Вместо аналитического вычисления матрицы Якоби левой части системы нелинейных уравнений можно задать его шаблон, т. е. расположение ненулевых элементов, в JacobPattern, тогда в процессе вычислений только ненулевые элементы матрицы Якоби будут аппроксимироваться конечными разностями, что значительно ускорит вычисления.

**Примеры**

Следующие разделы посвящены применению некоторых опций, управляющих вычислительным процессом, на примере решения большой системы нелинейных уравнений. Разобрано создание приложения с графическим интерфейсом пользователя, облегчающее использование возможностей Optimization Toolbox.

**Решение системы нелинейных уравнений**

Рассмотрим пример решения системы нелинейных уравнений, в котором задание аналитических выражений для элементов матрицы Якоби существенно уменьшает время вычислений. Необходимо решить приведенную ниже систему нелинейных уравнений F(x) = 0 для n = 1000.



Файл-функция largesys вычисляет значения компонент F2(x),..., Fn-1(x)

вектор-функции в цикле for (листинг 16.14).

Листинг 16.14. Файл-функция, вычисляющая левую часть системы уравнений (6.14)

function F = largesys(x)

n = length(х);

F = rand(n,1);

F(l) = 2\*х(1)л2 - х(2) - 1;

for i = 2:n-l

F(i) = -x(i - 1) + 2\*x(i)/N2 - x(i + 1) - 1;

end

F(n) = -x(n - 1) '+ 2\*x(n)"2 - 1;

Размерность задачи достаточно большая, и, как мы увидим, ее решение может занять значительное время, если не принимать во внимание разреженную структуру исходной системы. В данном примере вычисление матрицы Якоби дает существенный выигрыш во времени. Матрица Якоби решаемой системы



имеет достаточно простую структуру – она является сильно разреженной (трехдиагональной):



Напомним, что разреженные матрицы создаются при помощи функции sparse.

Очевидно, что матрица Якоби представляется суммой J = D + Dl + DT разреженных матриц D и D1, где



Дополните файл-функцию largesys нахождением разреженного представления для матрицы Якоби, которая вычисляется и возвращается во втором выходном аргументе, если файл-функция вызывается с двумя выходными аргументами. Используйте условный оператор if и переменную nargout для проверки числа выходных аргументов.

Текст модернизированной файл-функции largesysj приведен в листинге 16.15.

function [F, J] = largesysj(x)

% Файл-функция для вычисления левой части системы нелинейных уравнений

% и матрицы Якоби

% Вычисление компонент вектор-функции F

n = length(x);

F = rand(n,1);

F(1)= 2\*x(1)^2 - x(2) - 1;

for i = 2:n - 1

F(i) = -x(i - 1) + 2\*x(i)^2 - x(i + 1) - 1;

end

F(n) = -x(n - 1) + 2\*x(n)^2 - 1;

% Если число выходных аргументов более единицы, то требуется найти

% разреженное представление матрицы Якоби

if nargout > 1

% Матрица Якоби является суммой трех матриц J = D + D1 + D1'

% Формирование диагонали матрицы D

d = 4\*x;

% Инициализация разреженного представления для матрицы D

Diag = sparse(1:n, 1:n, d, n, n);

% Формирование вектора побочной диагонали

d2 = -ones(1, n - 1);

% Инициализация разреженного представления для матрицы D1

D1 = sparse(2:n, 1:n - 1, d2, n, n);

% Вычисление разреженной матрицы Якоби

J = Diag + D1 + D1';

end

Решение системы нелинейных уравнений оформите в файл-программе (листинг 16.16), в которой задаются число переменных и вектор начального приближения и используется fsolve для поиска корней. Наша цель состоит в исследовании эффективности применения явных формул для элементов матрицы Якоби, поэтому сначала обратитесь к fsolve со значением 'off' параметра Jacobian (оно установлено по умолчанию). Заметьте, что в этом Случае fsolve будет вызывать файл-функцию largesysj с одним выходным аргументом и матрица Якоби вычисляться не будет. Затем модифицируйте управляющую структуру, установив параметр Jacobian в значение в 'on', и снова примените fsolve. Для контроля за временем счета удобно задействовать встроенные функции tic и toc.

Листинг 16.16. Файл-программа (large.m) для решения большой системы нелинейных уравнений

n = 1000; % число переменных

x0 = ones(1, n); % начальное приближение

% Решение системы без использования явных формул

% для элементов матрицы Якоби

options = optimset('Display', 'iter', 'Diagnostics', 'on');

tic

x = fsolve(@largesysj, x0, options);

toc

% Решение системы с использованием явных формул

% для элементов матрицы Якоби

options = optimset(options, 'Jacobian', 'on');

tic

x = fsolve(@largesysj, x0, options);

toc

Выполнение операторов листинга 16.16 свидетельствует об эффективности использования явных формул для вычисления элементов матрицы Якоби по сравнению с аппроксимацией их конечными разностями – время счета отличается примерно в 50 раз. Поскольку параметр Diagnostics имеет значение 'on', то в командное окно сначала выводится информация о количестве неизвестных, способе вычисления частных производных и применяемом алгоритме (в данном случае– метод доверительной области). Работа fsolve сопровождается отображением сведений о каждом шаге вычислительного процесса (опция Display установлена в 'iter'). Сведения представлены в виде таблицы:

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Diagnostic Information

Number of variables: 1000

Functions

 Objective: largesysj

 Gradient: finite-differencing

Algorithm selected

 trust-region dogleg

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% End diagnostic information

 Norm of First-order Trust-region

 Iteration Func-count f(x) step optimality radius

 0 1001 998 3 1

 1 2002 871.767 1 2.38 1

 2 3003 567.03 2.5 2.13 2.5

 3 4004 36.9426 6.25 0.932 6.25

 4 5005 0.0498931 1.87805 0.0273 15.6

 5 6006 6.87178e-008 0.0643165 3e-005 15.6

 6 7007 1.34786e-019 7.56624e-005 4.11e-011 15.6

Optimization terminated: first-order optimality is less than options.TolFun.

Elapsed time is 3.127785 seconds.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Diagnostic Information

Number of variables: 1000

Functions

 Objective and gradient: largesysj

Algorithm selected

 trust-region dogleg

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% End diagnostic information

 Norm of First-order Trust-region

 Iteration Func-count f(x) step optimality radius

 0 1 998 3 1

 1 2 871.767 1 2.38 1

 2 3 567.03 2.5 2.13 2.5

 3 4 36.9426 6.25 0.932 6.25

 4 5 0.0498932 1.87805 0.0273 15.6

 5 6 6.87172e-008 0.0643166 3e-005 15.6

 6 7 1.31898e-019 7.56621e-005 4e-011 15.6

Optimization terminated: first-order optimality is less than options.TolFun.

Elapsed time is 0.263800 seconds.

>>

Столбик Iteration содержит номер итерации; Func-count – число вызовов функции; f(x) – сумма квадратов значений левых частей уравнений системы для текущего приближения; Norm of step – норма шага на текущей итерации; First-order optimality – бесконечная норма градиента, вычисленного для текущего приближения; Trust-region radius – радиус доверительной области.

Данный раздел описывает только решение системы нелинейных уравнений методом доверительных областей при помощи fsolve. В функции fsolve реализованы и другие алгоритмы решения, которые выбираются при помощи опций NonlEqnAlgorithm, LargeScale, LineSearchType управляющей структуры. Перед решением задач в Optimization Toolbox полезно обратиться к информации о настройках алгоритмов выбранной функции. Эти сведения могут быть почерпнуты из справочной системы Toolbox. Функции и допустимые параметры для них описаны в разделе Optimization Toolbox: Function Reference, задание параметров – в Optimization Toolbox: Function Reference: Optimization Parameters, а сами алгоритмы – в Optimization Toolbox: Standard Algorithms и Optimization Toolbox: Large-Scale Algorithms.

Для дальнейшего усвоения материала рекомендуется также изучить примеры решения оптимизационных задач, приведенные в справочной системе по Optimization Toolbox.

Ануфриев И. Е., Смирнов А. Б., Смирнова Е. Н. MATLAB 7. – СПб.: БХВ-Петербург, 2005. – 1104 с: ил.