АННОТАЦИЯ

В работе предлагаются легко реализуемые численные методы приближённого поиска глобального минимума непрерывной на отрезке функции и непрерывной на брусе функции двух переменных, на которые не накладываются никакие дополнительные условия типа Липшицевости. Единственным требованием к функции является её непрерывность на выпуклом компакте (отрезке или прямоугольнике). Приводятся результаты численных экспериментов. Методы могут быть полезны при решении задач негладкого математического программирования.

ANNOTATION

The paper proposes an easily implemented numerical methods for the approximate search for a global minimum of a continuous function on an interval and of a continuous function of two variables on a bar without any additional constraints such as conditions of Lipschitz type. The only requirement for the function is its continuity on a convex set (interval or rectangle). The results of numerical experiments are given. Methods can be useful in solving problems of non-smooth mathematical programming.

ВВЕДЕНИЕ

В работе [1] было введено понятие ε-Липшицевости:

Пусть функция определена на компактном множестве из нормированного пространства . Она называется ε-Липшицевой, если выполняется следующее условие:

(1)

Там же было доказано, что выполнение условия (1) необходимо и достаточно для непрерывности функции на . Оказалось, что свойство ε-Липшицевости функции можно с успехом применять к созданию методов оптимизации негладких(может быть существенно негладких) функций. Так, в указанной работе [1], была предложена модификация метода Пиявского (метода ломаных) поиска приближённого минимума ε-Липшицевой функции. Таким образом, открывается возможность минимизации непрерывных на компакте функций без добавления каких-либо дополнительных условий (кроме непрерывности) на минимизируемую функцию.

Следует заметить, что из определения (1) не следует положительности постоянной , которую будем называть ε-константой Липшица.

Для дальнейших выкладок для нас будет существенной её строгая положительность. В [2] было показано, что неравенство при заданном выполняется, если найдутся такие, что

(2)

Условие (2) было названо согласованностью функции и . Легко понять, что это условие не сильно сужает класс рассматриваемых функций, и мы будем всюду в дальнейшем полагать это условие выполненным.

Свойство ε-Липшицевости вызвало появление ряда работ [3, 4, 5, 6], в каждой из которых приходилось решать вспомогательную задачу минимизации ε-Липшицевой функции. При этом предполагалось, что значение оценки было известно или известна функциональная зависимость от ε. Отыскание указанных зависимостей является непростой задачей. Можно сказать, что непростой задачей является уже задача нахождения постоянной Липшица для Липшицевых функций, задача же нахождения постоянной является задачей ещё более сложной. В последнее время появилась работа [2], в которой предложен один из возможных алгоритмов приближённого отыскания постоянной по заданному . В данной работе считается, что эта задача решена и нам известна минимальная оценка .

# Глава 1. Алгоритм равномерного перебора отыскания глобального минимума функции одной переменной непрерывной (ε-Липшицевой) на отрезке

## Описание алгоритма и его обоснование

Здесь мы сформулируем и обоснуем алгоритм минимизации ε-Липшицевой функции на отрезке.

Введём параметры алгоритма:

* ε – параметр, выбираемый из условия (1)
* – погрешность, с которой отыскивается приближённое значение минимума функции

Всюду предполагается выполнение неравенства .

Обозначим:

* - отрезок, на котором отыскивается минимум
* - наименьшее значение функции на отрезке (которое, очевидно, существует)
* Положим , где n-количество отрезков, на которые разбивается отрезок .

Потребуем выполнение неравенства , отсюда . Полученное условие позволяет найти количество пробных точек.

Построим последовательность пробных точек отрезка по следующему правилу:

Нашей задачей будет показать, что или, что то же самое, .

Очевидно, для этого будет достаточно показать, что среди членов последовательности найдётся точка , для которой выполняется условие:

(3)

Обозначим и пусть . Тогда из неравенства (1) следует

или, что то же самое

(4)

Но поскольку , то выполняется следующее неравенство

Тогда из неравенства (4) следует:

(5)

Пусть - точка глобального минимума функции на отрезке . Очевидно, найдётся отрезок такой, что .

Тогда, из неравенства (5) следует:

Что и требовалось доказать.

## Замечание к алгоритму

Использование алгоритма предполагает априорное знание ε-константы Липшица для заданного ε.

Как видно из цитируемой литературы - лишь для некоторых элементарных функций удалось построить функциональную зависимость от ε. Последняя зависимость, из известных автору, получена в работе [6] для функции , что, как следует из работы, было сопряжено с трудностями. Однако, если дополнить приведённый алгоритм предварительным нахождением минимальной оценки ε-константы Липшица с помощью алгоритма, предложенного в [2], то указанная трудность снимается (правда ценой увеличения времени нахождения глобального минимума).

## Численные примеры

В качестве тестовой задачи приближённого нахождения глобального минимума функции была рассмотрена функция вида

для которой в [5] была получена оценка

Для двух различных наборов постоянных приведём результаты численного эксперимента.

*Таблица 1.Результаты работы алгоритма*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Отрезок | ε\* | ε |  |  |  |  |  |  | , *м*с |  |
| Набор 1. *b*1= *b*2= *b*3= 0; *a*1=–4, *a*2=–1, *a*3= 3 | | | | | | | | | |  |
|  |
|  | 0.01 | 0.005 | 50 | 0.0001 | –0.999999 | | 0.000001 | 99999 | 4 |  |
|  | 0.001 | 250 | 0.000036 | –1.000004 | | 0.002000 | 277777 | 11 |  |
| [–5; 5] |  |
| 0.001 | 0.0005 | 500 | 0.000001 | –3.999999 | | 0.000012 | 9999999 | 398 |  |
|  |  |
|  | 0.0001 | 2500 | 0.0000004 | –1.000000 | | 0.000202 | 27777777 | 1101 |  |
|  | 0.01 | 0.005 | 50 | 0.000100 | 2.999999 | | 0.000003 | 200000 | 9 |  |
|  | 0.001 | 250 | 0.000036 | –1.000000 | | 0.000007 | 555555 | 35 |  |
| [–10; 10] |  |
| 0.001 | 0.0005 | 500 | 0.000001 | 2.999999 | | 0.000026 | 20000000 | 751 |  |
|  |  |
|  | 0.0001 | 2500 | 0.0000004 | –1.000000 | | 0.000081 | 55555555 | 2059 |  |
|  | 0.01 | 0.005 | 50 | 0.000100 | 2.999999 | | 0.000004 | 300000 | 12 |  |
|  | 0.001 | 250 | 0.000036 | 2.999999 | | 0.000009 | 833333 | 32 |  |
| [–15; 15] |  |
| 0.001 | 0.0005 | 500 | 0.000001 | 2.999999 | | 0.000066 | 30000000 | 1148 |  |
|  |  |
|  | 0.0001 | 2500 | 0.0000004 | 2.999999 | | 0.000110 | 83333333 | 3059 |  |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Набор 2. *b*1=–1, *b*2=–1.005, *b*3= 0.5; *a*1=–4, *a*2=–1, *a*3= 3 | | | | | | | | |  |
|  |
|  | 0.01 | 0.005 | 50 | 0.000100 | –0.999999 | –1.004999 | 99999 | 3 |  |
| [–5; 5] | 0.001 | 250 | 0.000036 | –1.000004 | –1.003000 | 277777 | 9 |  |
|  |
| 0.001 | 0.0005 | 500 | 0.000001 | –0.999999 | –1.004982 | 9999999 | 367 |  |
|  |  |
|  | 0.0001 | 2500 | 0.0000004 | –1.000000 | –1.004798 | 27777777 | 997 |  |
|  | 0.01 | 0.005 | 50 | 0.000100 | –1.000000 | –1.004997 | 200000 | 7 |  |
|  | 0.001 | 250 | 0.000036 | –1.000000 | –1.004993 | 555555 | 23 |  |
| [–10; 10] |  |
| 0.001 | 0.0005 | 500 | 0.000001 | –1.000000 | –1.004973 | 20000000 | 711 |  |
|  |  |
|  | 0.0001 | 2500 | 0.0000004 | –1.000000 | –1.004919 | 55555555 | 1971 |  |
|  | 0.01 | 0.005 | 50 | 0.000100 | –1.000000 | –1.004995 | 300000 | 10 |  |
|  | 0.001 | 250 | 0.000036 | –0.999996 | –1.003000 | 833333 | 43 |  |
| [–15; 15] |  |
| 0.001 | 0.0005 | 500 | 0.000001 | –1.000000 | –1.004933 | 30000000 | 1056 |  |
|  |  |
|  | 0.0001 | 2500 | 0.0000004 | –1.000000 | –1.004834 | 83333333 | 2871 |  |

Сравнивая с результатами, полученными в [5], мы видим, что при более простой реализации алгоритма требуется больше пробных точек, однако время, необходимое на выполнение алгоритма, значительно уменьшается.

# Глава 2. Случай функции двух переменных

Далее рассмотрим случай применения метода равномерного перебора поиска глобального минимума двумерной ε-Липшицевой функции.

Дана функция такая, что . Область, на которой будет производиться изучение функции на минимум, разделим сеткой размера . Введём следующие обозначения:

* ε – параметр, выбираемый из условия (1)
* – погрешность, с которой отыскивается приближённое значение минимума функции
* – шаг алгоритма по оси ;
* – шаг алгоритма по оси ;
* - наименьшее значение функции на области (которое, очевидно, существует);
* - узел сетки, представляющий ячейку , где .

Определим значение . Для этого, сначала необходимо ограничить значение шага , допустим, следующим образом:

Тогда, очевидно

И, преобразовав это неравенство, получаем нижнюю границу шага по оси :

Аналогичным образом получаем оценку числа разбиений по оси :

Строго говоря, константы Липшица по оси и могут различаться, но мы можем взять большую из них и использовать как универсальную

И тогда величина шага по обеим осям станет одинаковой

Нашей задачей будет доказать истинность следующего неравенства:

Найдётся такой, что

(2.1)

(смотри рисунок 1)



*Рисунок 1 Геометрическое пояснение неравенства (2.1)*

Или, что то же самое

В качестве нормы удобно будет выбрать покоординатную норму

Тогда:

Так как соседние узлы сетки не могут располагаться друг от друга дальше, чем на величину, то

А так как и , и ограничены сверху одной величиной , тогда можно записать

Переписав, получаем требуемое неравенство

Что и требовалось доказать.

## Численные примеры

# Глава 3. Программная реализация алгоритмов

## Обоснование выбора языка и среды программирования

Программная реализация алгоритмов была выполнена на языке программирования C#, поскольку это отлично зарекомендовавший себя на рынке во всём мире язык со множеством удобного функционала, работающего «из коробки». К числу его положительных черт можно отнести:

* это объектно-ориентированный, мультипарадигмальный, рефлективный язык программирования
* C# является «родным» языком программирования для среды Microsoft .Net, потому что он самым тесным и эффективным образом интегрирован с ней
* При компилировании программы CIL(Common Intermediate Language - «высокоуровневый ассемблер» виртуальной машины [.NET](https://ru.wikipedia.org/wiki/Microsoft_.NET)) – инструкции распространяются в специальных единицах – сборках, и сопровождаются своими метаданными, что делает её самодостаточным объектом. Благодаря тому, что в исполняемом файле, помимо самой программы, находятся метаданные, можно получать объекты, которые описывают типы, модули и сборки во время исполнения кода (run-time).
* Автоматическое управление памятью;
* Библиотека .NET Framework содержит богатый набор классов для работы с различными видами данными, такими как: ввод / вывод информации, работа с базами данным, работа с сетью, работа с графикой и так далее.
* Для C# кода предоставляется огромный набор «синтаксического сахара», позволяющего создавать лаконичный, понятный для чтения код. Также, сокращая код, необходимый для реализации одних и тех же действий, уменьшается шанс допустить ошибку. Примерами «синтаксического сахара» являются: лямбда выражения, анонимные типы, LINQ (набор появившихся в .NET функций, которые значительно расширяют возможности синтаксиса языков C# и Visual Basic).

В качестве среды программирования была выбрана MS Visual Studio 2017 Community, как последняя версия Visual Studio на момент начала разработки. Community версия, к тому же, распространяется абсолютно бесплатно и при этом сохраняет весь свой основной функционал от старших версий.

# Глава 4. Тестирование программного продукта

## Основные понятия и принципы тестирования программного продукта

Тестирование программного любого продукта, вне зависимости от его назначения или объёма исходного кода, является важной частью разработки программного продукта. Без полноценно проведённого тестирования невозможно оценить, насколько качественно программный продукт выполнен. Чем качественнее будут проведены тесты, тем меньше вероятность, что будут упущены ошибки, которые в будущем придётся исправлять

Тестирование — это процесс испытания, исследования [программного продукта](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), имеющий следующие основные цели:

* показать, что программный продукт выполнен качественно и в соответствии со всеми требованиями;
* обнаружить такие ситуации, при возникновении которых поведение программы является некорректным, нежелательным или вовсе непредсказуемым.

Информационные потоки процесса тестирования изображены на рисунке 2.

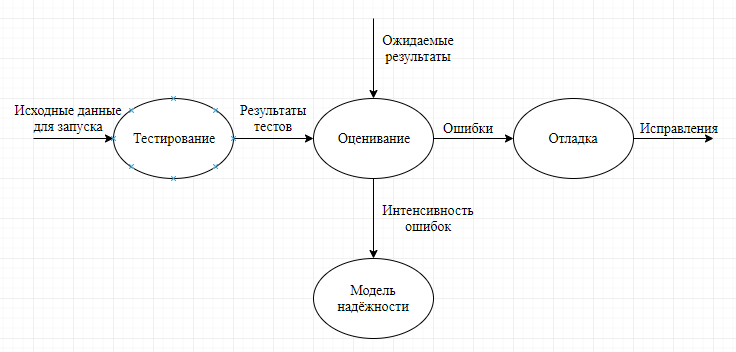


Рисунок 2 Информационные потоки процесса тестирования

В тех случаях, когда программный продукт разрабатывается для массового применения, отсутствие должного тестирования, а в следствии, упущение ряда ошибок, может вызвать негативное отношение к нему среди пользователей.

Сколь хорошо не были бы проведены тесты, их результаты не могут однозначно указывать на полное отсутствие возможности возникновения ошибок. Для наиболее полного определения качества программного обеспечения производится анализ совокупности следующих составляющих:

* [надёжность](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C);
* [сопровождаемость](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BF%D1%80%D0%BE%D0%B2%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F);
* практичность;
* эффективность;
* мобильность;
* функциональность.

В процессе тестирования для каждого модуля программного обеспечения создаются наборы тестов с вариациями исходных значений и ожидаемыми результатами отработки модулей.

По выполнению каждого теста результаты их исполнения проверяются или, другими словами, реальные результаты исполнения тестов сравниваются с ожидаемыми, заранее подготовленными и проверенными результатами. При обнаружении несовпадений, фиксируется ошибка. Каждая найденная ошибка обязательно должна быть исправлена, поэтому начинается процесс поиска причин неверной работы модулей программного продукта и его отладка. На поиск места дефекта и исправление может потребоваться от пары минут до нескольких дней. Эта неопределенность приводит к затруднению планирования действий.

По окончании отладки на исправленные места исходных кодов пишутся новые, либо исправляются старые тесты и процесс проверки начинается заново.

Если функции программного обеспечения реализованы правильно, а обнаруженные ошибки легко исправляются, может быть сделан один из двух выводов [8]:

* качество и надежность ПО удовлетворительны;
* тесты не способны обнаруживать серьезные ошибки.

Существуют 2 принципа тестирования программы:

* функциональное тестирование (тестирование «черного ящика»);
* структурное тестирование (тестирование «белого ящика»).