Чигарев Дмитрий 381807-1

Лабораторная работа

«Оптимизация многомерных функций численными методами»

Целью данной лабораторной работы является решение задачи минимизации 4х заданных многомерных функций с заданной точностью. В качестве механизма решения, предлагается выбрать три итерационных метода оптимизации и сравнить полученные результаты.

Исходные данные

4 задачи предложенные к решению:

1. $\begin{cases} Q(x_1, x_2) = 100(x_1^2 + x_2)^2 + (1 - x_1)^2 \to min \\ X = (x_1, x_2) \in R^2 \\ x^0 = (-1.2, 1) \\ \varepsilon = 10^{-7} \end{cases}$

2. $\begin{cases} Q(x_1, x_2) = 100(x_2 + x_1^3)^2 + (1 - x_1)^2 \to min \\ X = (x_1, x_2) \in R^2 \\ x^0 = (-1.2, -1) \\ \varepsilon = 10^{-7} \end{cases}$

3. $\begin{cases} Q(x_1, x_2) = 100(x_2 + x_1^3)^2 + (1 - x_1)^2 \to min \\ X = \{(x_1, x_2) \mid x_1 \in [-1.2, 1],\ x_2 \in [-1, 1]\} \\ x^0 = (-1.2, -1) \\ \varepsilon = 10^{-7} \end{cases}$

4. $\begin{cases} Q(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4 \to min \\ X = (x_1, x_2, x_3, x_4) \in R^4 \\ x^0 = (3, -1, 0, 1) \\ \varepsilon = 10^{-7} \end{cases}$

Каждая из функций представляет из себя сумму неких величин, возведенных в четные степени. Отсюда становится понятно, что минимальное значение ограничено снизу нулем. Посмотрев на предложенные функции ещё пару секунд, нам становится очевидно, что минимальные значения этих функций и есть ноль, и даже становятся видны точки, при которых это значение достигается.

Посмотрим, получится ли у рассматриваемых методов так же без труда определить оптимум.

Метод Хука-Дживса

Метод Хука-Дживса – итерационный метод, который на каждом шаге своей работы смешается в сторону локального экстремума функции, причем без использования градиента. Для этого он покомпонентно формирует смещение в сторону уменьшения значения функции, и затем двигается в этом направлении, пока убывание функции не сменится её ростом.

Если говорить более формально, то алгоритм записывается следующим образом:

1. Если требуемая точность достигнута, то выход, иначе шаг 2
2. Дана текущая точка x^k
3. Для каждой компоненты x^k:
   3.1 Изменить компоненту: $x_i^{k'} = x_i^k + h'$
   3.2 Если значение функции уменьшилось, то шаг 3, иначе 3.3

3.3 Если $|h'| < eps$ И $h' < 0$, то $x_i^{k\prime} = x_i^k$ и шаг 3
Если $|h'| \geq eps$, то $h' = h'/reducer$, иначе $h' = -h$
Шаг 3.1

4. $h' = h$
   Направление смещения: $dir = x^k - x^{k\prime}$
5. Если $h' < eps$, то шаг 1, иначе шаг 6
6. Если $f(x^{k+1} + h' * dir) < f(x^{k+1})$, то $x^{k+1} = x^{k+1} + h' * dir$
   Иначе $h' = h'/reducer$
   Шаг 5

Как видно, у метода есть два параметра:

1. h – максимально возможный коэффициент шага метода.
2. $reducer$ – коэффициент уменьшения шага, в случае “перепрыгивания” точки минимума.

оба они влияют на точность и продолжительность поиска оптимального шага в сторону убывания.

Формально один шаг алгоритм разделют на две фазы: исследующий поиск (шаг 3) – формирование направления убывания функции, и поиск по образцу (шаги 5, 6) – движение в направлении убывания функции, пока убывание не сменится ростом, после того как найденное направление исчерпало себя, производится поиск нового направления.

Как видно из алгоритма, метод способен искать лишь безусловные экстремумы, однако в нашем списке задач есть одна, в которой область поиска задана ограниченным множеством, т.е. ставится задача поиска условного экстремума. Разрешим эту ситуацию следующим образом: каждый раз, когда методу нужно взять следующую точку, вместо неё он будет брать проекцию этой точки на допустимое множество.

Реализуем описанный выше алгоритм в мат. пакете SciLab и посмотрим на результат. Параметры для каждой задачи будут такими:

1. $h = 0.5,\ reducer = 2$
2. $h = 1,\ reducer = 1.2$
3. $h = 1,\ reducer = 1.2$
4. $h = 0.5,\ reducer = 1.5$

Критерий останова будет следующий:

$$\left|F\left(x^k\right) - F\left(x^{k-1}\right)\right| < \varepsilon \ \ \& \ \ \left|x^k - x^{k-1}\right| < \varepsilon$$

Если оба условия выполнены – метод считает, что требуемая точность достигнута и прекращает свою работу.

```
============================ Метод Хука-Дживса ============================

--------------- Problem 1 ---------------
100(x^2 - y)^2 + (1-x)^2 -> min | X0 = (-1.2, 1) | eps = 10e-7
Found solution in 55 iterations:
       Result point: [9.999989e-01, 9.999981e-01]
       Function value: 1.586149e-11

--------------- Problem 2 ---------------
100(y - x^3)^2 + (1-x)^2 -> min | X0 = (-1.2, -1) | eps = 10e-7
Found solution in 46 iterations:
       Result point: [1.000052e+00, 1.000155e+00]
       Function value: 2.705273e-09

--------------- Problem 3 ---------------
```

```
100(y - x^3)^2 + (1-x)^2 -> min | X0 = (-1.2, -1) | eps = 10e-7 | x ∈ [-1.2, 1] | y ∈ [-1, 1]
Found solution in 40 iterations:
       Result point: [1.000000e+00, 1.000000e+00]
       Function value: 0.000000e+00

--------------- Problem 4 ---------------
(x1 + 10x2)^2 + 5(x3 - x4)^2 + (x2 - 2x3)^4 + 10(x1 - x4)^4 | X0 = (3, -1, 0, 1) | eps = 10e-7
Found solution in 14 iterations:
       Result point: [5.463847e-03, -5.464233e-04, 2.760807e-03, 2.760669e-03]
       Function value: 1.889983e-09
```

Точное решение удалось найти лишь для третьей задачи, интересно, что для второй, которая отличается от третей лишь отсутствием ограничения на зону поиска, точное решение не нашлось.

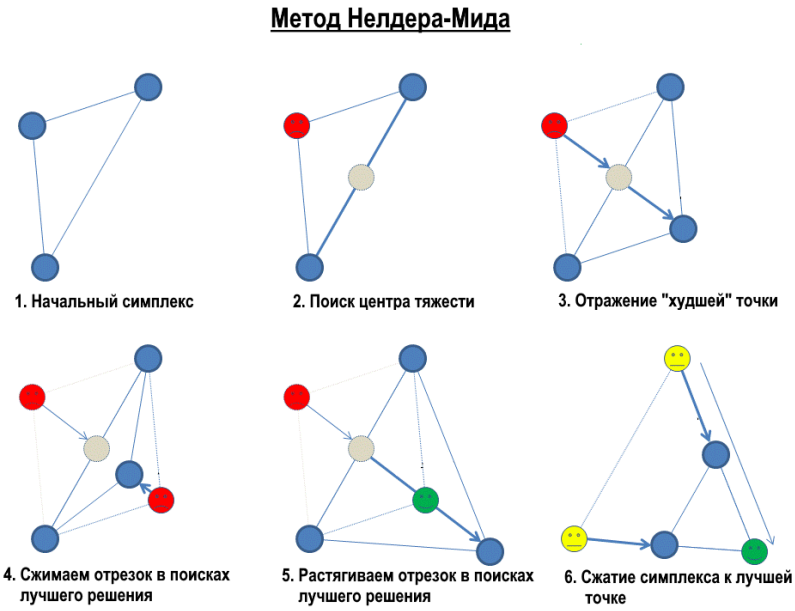
Каждое из решений удовлетворяет условию $|F(x) - F(x^*)| < \varepsilon$, но ни одно (кроме третьего) не удовлетворяет $|x - x^*| < \varepsilon$.

Метод Нелдера-Мида

Метод Нелдера-Мида – итерационный метод оптимизации функции нескольких переменных, без использования градиента.

В отличии от многих других методов оптимизации, объектом манипуляций в методе Нелдера-Мида является не точка пространства, а его симплекс (т.е. $n + 1$ точка пространства). Суть метода заключается в сравнении значения функций в $n + 1$ точках, задающих симплекс, и последующей деформации симплекса в направлении локального оптимума, посредством четырех операций:

1. Отражение худшей точки симплекса, относительно точек с лучшими значениями функции.
2. Сжатие отрезка, задающего отражение худшей точки.
3. Растяжение отрезка, задающего отражение худшей точки.
4. Сжатие симплекса к лучшей его точке.



Ровно по количеству возможных преобразований у метода есть 4 параметра:

1. **Коэффициент отражения** (a, классическое значение = 1) – насколько “симметрично” будет отражаться худшая точка (x_h), относительно центра масс (x_c) других точек: $x_r =$

$(1+a)x_c - ax_h$. При $a = 1$ будет получаться симметрия, этот параметр и используется чаще всего.

2. **Коэффициент сжатия** (b, как правило меньше единицы, классическое значение = 0.5) – регулирует во сколько раз, производить сжатие отрезка, соединяющего худшую точку и её отражение: $x_s = bx_r + (1-b)x_c$
3. **Коэффициент растяжения** (c, как правило больше единицы, классическое значение = 2) – регулирует во сколько раз производить растяжение отрезка, соединяющего худшую точку и её отражение: $x_s = cx_r + (1-c)x_c$
4. **Коэффициент сжатия симплекса** (d, как правило меньше единицы, классическое значение = 0.5) – регулирует во сколько раз уменьшать расстояния между лучшей точкой симплекса (x_b) и всеми остальными: $x_i' = x_b + (x_i - x_b)d$

(Во всех приведенных ниже запусках использовались “классические” значения параметров.)

Поскольку метод на каждом своем шаге модифицирует не одну точку, а симплекс, то оптимальная точка может оставаться неизменной на протяжении нескольких итераций подряд, хотя метод быть может ещё не сошелся, и активно модифицирует другие вершины симплекса. Соответственно, предыдущий критерий останова, оценивающий разницу между текущей и предыдущей точкой, использовать мы не можем. Вместо разницы между точками будем оценивать разницу между симплексами:

$$\left|simplex^k - simplex^{k-1}\right| < \varepsilon$$

Если данное условие выполнено – метод считает, что требуемая точность достигнута, либо площадь рассматриваемого симплекса стала слишком мала, и прекращает работу.

```
============================ Метод Нелдера-Мида ============================

--------------- Problem 1 ---------------
100(x^2 - y)^2 + (1-x)^2 -> min | X0 = [(-1.2, 1), (3, -2), (0, 0)] | eps = 10e-7
Found solution in 101 iterations:
       Result point: [1.000000e+00, 1.000000e+00]
       Function value: 2.559770e-14

--------------- Problem 2 ---------------
100(y - x^3)^2 + (1-x)^2 -> min | X0 = [(-1.2, -1), (-3, 2), (0, 0)] | eps = 10e-7
Found solution in 86 iterations:
       Result point: [1.000000e+00, 1.000000e+00]
       Function value: 7.019389e-14

--------------- Problem 3 ---------------
100(y - x^3)^2 + (1-x)^2 -> min | X0 = [(-1.2, -1), (-1.2, 1), (0, 0)] | eps = 10e-7 | x ∈
[-1.2, 1] | y ∈ [-1, 1]
Found solution in 24 iterations:
       Result point: [1.000000e+00, 1.000000e+00]
       Function value: 0.000000e+00

--------------- Problem 4 ---------------
(x1 + 10x2)^2 + 5(x3 - x4)^2 + (x2 - 2x3)^4 + 10(x1 - x4)^4 | X0 = [(3, -1, 0, 1), (-3, 1,
0, -1), (0, 0, 1, 2), (1, 2, 0, 0), (1, 4, 8, 8)] | eps = 10e-7
Found solution in 343 iterations:
       Result point: [4.059178e-07, -4.059141e-08, 8.513076e-07, 8.513091e-07]
       Function value: 3.586171e-23
```

Как видно, метод за приемлемое кол-во итераций смог найти решение, удовлетворяющее заданной точности, для каждой из представленных задач. Причем в отличии от метода Хука-Дживса, для всех полученных решений помимо условия $|F(x) - F(x^*)| < \varepsilon$ выполняется ещё и более сильное: $|x - x^*| < \varepsilon$.

Метод градиентного спуска

Метод градиентного – итерационный метод, который на каждом шаге своей работы смешается в сторону локального экстремума, двигаясь вдоль градиента функции. В случае поиска минимума, мы должны двигаться в направлении противоположном градиенту в текущей точке.

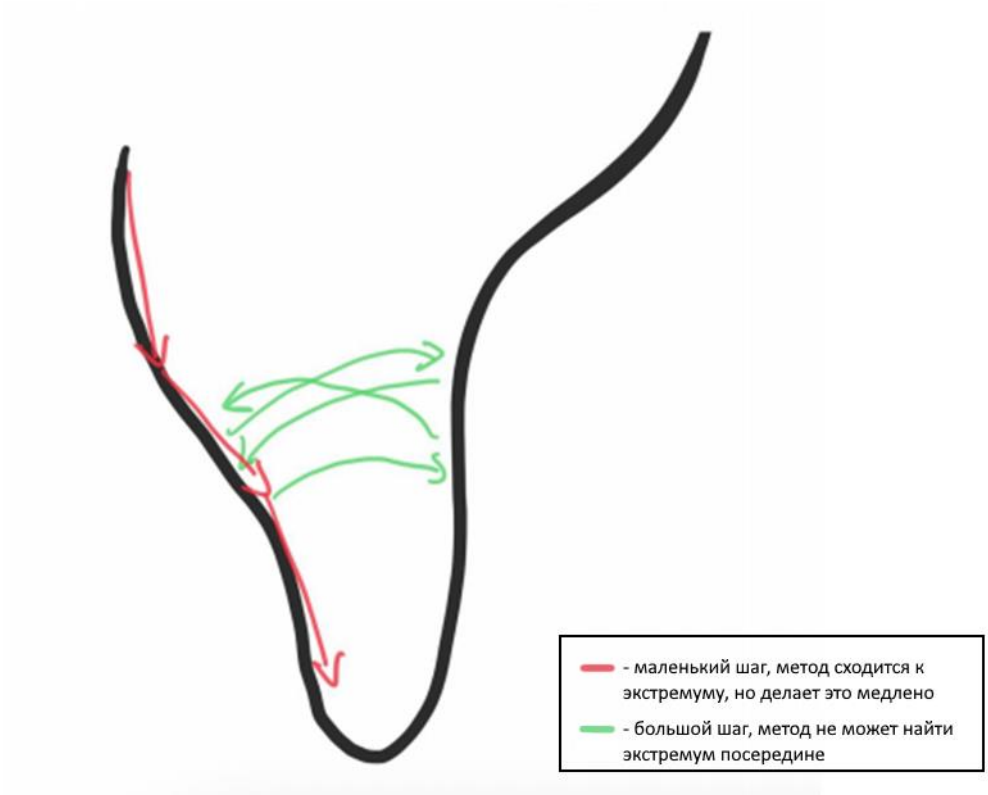
Формальный алгоритм:

1. Если требуемая точность достигнута, то выход, иначе шаг 2
2. Дана текущая точка x^k
3. $x^{k+1} = x^k - h^k * grad(f, x^k)$
4. Шаг 1

Описанный алгоритм не предоставляет инструкций по правильному выбору параметра h^k и дает повод к порождению множества модификаций этого метода, которые отличаются лишь правилами выбора h^k.

Существует два основных вида модификаций градиентного метода: с постоянным шагом и с переменным.

Как следует из названия, в методах с постоянным шагом, коэффициент h фиксируется и не изменяется на протяжении всего метода. Преимущество такого подхода в его простоте – на каждой итерации требуется лишь подсчет градиента, однако в решении большинства задач такой подход терпит неудачу. Дело в том, что на начальных итерациях, когда мы ещё далеко от экстремума, нам выгодно делать большие шаги, для проскакивания локальных экстремумов и в целом более быстрого приближения к искомой точке. Когда же мы уже находимся в окрестности искомой точки, нам выгодно иметь маленький шаг, чтобы не уйти за пределы окрестности. Оказывается, что на реальных задачах подобрать один шаг на все случаи жизни зачастую не получается: метод либо сходится очень долго или застревает в первом же локальном минимуме (при маленьком шаге), либо вовсе расходится (при большом шаге):



Изначально, была надежда, что хотя бы для одной из 4х представленных задач удастся подобрать такой статичный шаг, при котором бы метод сошелся менее чем за 10000 итераций, но все попытки оказались тщетны.

Частично решают описанные проблемы методы с переменным шагом. На каждой итерации, параметр h подбирается индивидуально исходя из значения градиента и стадии сходимости. Метод, который использовался для решения задач – метод наискорейшего спуска. Его принцип похож на стадию поиска по образцу из метода Хука-Дживса – на каждом шаге градиентного спуска для параметра h ставится задача одномерной оптимизации:

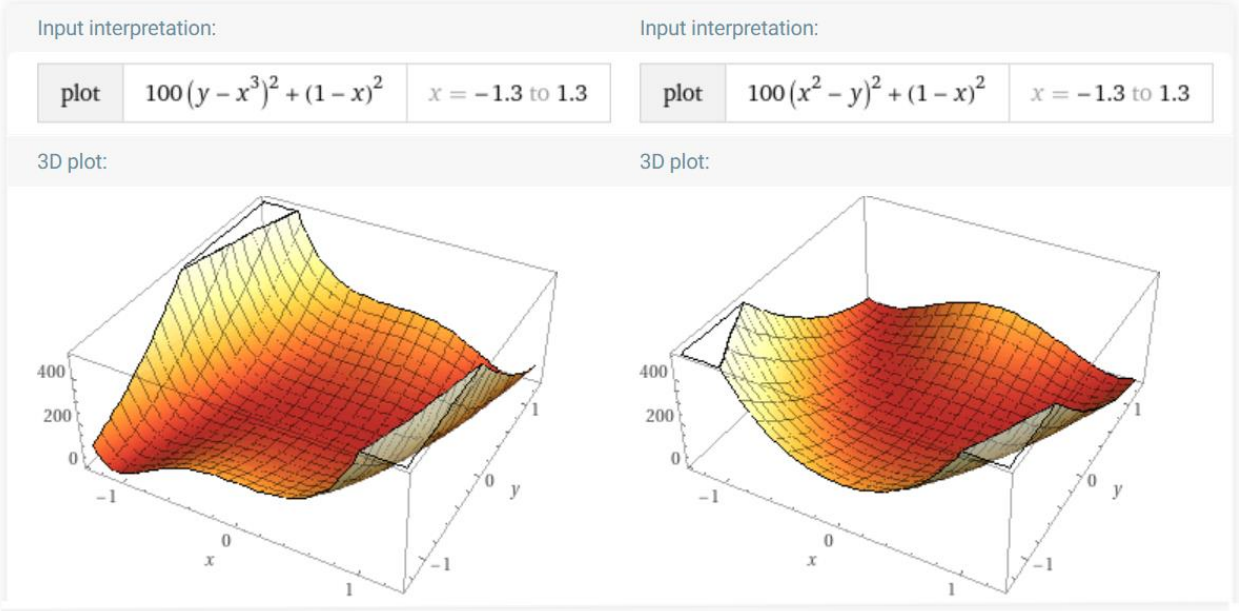
$$h = arg \min_{h \epsilon [0,M]} f(x^k - h * grad(f, x^k))$$

Решив её, получим оптимальное значение параметра на данном шаге. Как правило, решение этой задачи не слишком трудоемко, что позволяет решать её в цикле основного метода без особых затрат. В качестве метода одномерной оптимизации был выбран симметричный метод золотого сечения, он обеспечивает сверх-линейную сходимость.

Реализуем описанный выше алгоритм в мат. пакете SciLab. Для имитации поиска условного экстремума в третьей задачи, также как и в двух предыдущих методах был использован метод проекции на допустимое множество. Условие остановки такое же, как и в методе Хука-Дживса.

```
====================== Градиентный метод наискорейшего спуска ======================

--------------- Problem 1 ---------------
100(x^2 - y)^2 + (1-x)^2 -> min | X0 = (-1.2, 1) | eps = 10e-7
Found solution in 161 iterations:
      Result point: [1.000024e+00, 1.000047e+00]
      Function value: 5.569881e-10

--------------- Problem 2 ---------------
100(y - x^3)^2 + (1-x)^2 -> min | X0 = (-1.2, -1) | eps = 10e-7
Found solution in 903 iterations:
      Result point: [9.998397e-01, 9.995187e-01]
      Function value: 2.571737e-08

--------------- Problem 3 ---------------
100(y - x^3)^2 + (1-x)^2 -> min | X0 = (-1.2, -1) | eps = 10e-7 | x ∈ [-1.2, 1] | y
∈ [-1, 1]
Found solution in 5 iterations:
      Result point: [1.000000e+00, 1.000000e+00]
      Function value: 0.000000e+00

--------------- Problem 4 ---------------
(x1 + 10x2)^2 + 5(x3 - x4)^2 + (x2 - 2x3)^4 + 10(x1 - x4)^4 | X0 = (3, -1, 0, 1) |
eps = 10e-7
Found solution in 6216 iterations:
      Result point: [1.921651e-02, -1.921426e-03, 9.573861e-03, 9.579394e-03]
      Function value: 2.834693e-07
```

Можно сказать, что представленные задачи оказались не по зубам градиентному методу: для каждой задачи требуется огромное кол-во итераций, и даже так, метод находит решения хуже, чем два предыдущих. Подсказку почему так произошло дает третья задача – стоило нам ограничить зону поиска, как метод тут же сошелся за приемлемое время. Дело в том, что за пределами этой области рассматриваемые функции резко взмывают вверх.



Если метод случайно попадет эту зону взрывного роста значения функции, то спуск от туда займет очень большое кол-во итераций.

Ограничим область определения функции из первой задачи и посмотрим какие изменения в поведении мы получим:

```
====================== Градиентный метод наискорейшего спуска ======================

--------------- Problem 1 (with limitations) ---------------
100(x^2 - y)^2 + (1-x)^2 -> min | X0 = (-1.2, 1) | eps = 10e-7 | x ∈ [-1.2, 1] | y
∈ [-1, 1]
Step 1: xk = [1.000000e+00, 1.000000e+00]| F(xk) = 0.000000e+00
Step 2: xk = [1.000000e+00, 1.000000e+00]| F(xk) = 6.524230e-22
Found solution in 2 iterations:
      Result point: [1.000000e+00, 1.000000e+00]
      Function value: 6.524230e-22
```

Метод тут же сошелся! Отсюда можно сделать вывод о том, что перед тем, как применять градиентный метод, следует сначала исследовать поведение производной, и в случае необходимости ограничить область поиска от “плохих” зон.

Приложения

1. Исходный код программы: https://github.com/proxodilka/optimization-analysis-labs