Оптимальные методы первого порядка

Подготовили студенты 691 группы:

Бармакова Азалия, Воропаев Роберт.

Преподаватель: Тренин Сергей Алексеевич.

Теоретическая часть

Основные определения

Решение многих теоретических и практических задач сводится к отысканию экстремума скалярной функции $f(\mathrm{x})$ n-мерного векторного аргумента x. В дальнейшем под x будем понимать вектор-столбец (точку в n-мерном пространстве). Оптимизируемую функцию $f(x)$ называют целевой функцией.

В дальнейшем будем говорить о поиске минимального значения функции $f(x)$ записывать эту задачу следующим образом: $f(x) \to min$.

Вектор x^*, определяющий минимум целевой функции, называют оптимальным. Решением задачи (оптимальной точкой) называют допустимую точку x^*, в которой целевая функция $f(\mathrm{x})$ достигает своего минимального значения.

В данном проекте мы рассматриваем задачу безусловной оптимизации выпуклой функции многих переменных.

Классификация методов

Возможны два подхода к решению задачи отыскания минимума функции многих переменных $f(x) = f(x_1, \ldots, x_n)$ при отсутствии ограничений на диапазон изменения неизвестных. Первый подход лежит в основе косвенных методов оптимизации и сводит решение задачи оптимизации к решению системы нелинейных уравнений, являющихся следствием условий экстремума функции многих переменных. Как известно, эти условия определяют, что в точке экстремума x^* все первые производные функции по независимым переменным равны нулю:

$$\frac{df}{dx_i} = 0, i = 1, \ldots, n.$$

Эти условия образуют систему n нелинейных уравнений, среди решений которой находятся точки минимума.

Решение систем нелинейных уравнений - задача весьма сложная и трудоемкая. Вследствие этого на практике используют второй подход к минимизации функций, составляющий основу прямых методов. Суть их состоит в построении последовательности векторов $x_0, x_1, \ldots, x_n,$ таких, что $f(x_0) > f(x_1) > f(x_n) > \ldots$. В качестве начальной точки x_0 может быть выбрана произвольная точка, однако стремятся использовать всю имеющуюся информацию о поведении функции $f(x)$, чтобы точка x_0 располагалась как можно ближе к точке минимума. Переход (итерация) от точки x_k к точке $x_{k+1}, k = 0, 1, 2, \ldots$, состоит из двух этапов:

1. выбор направления движения из точки x_k;
2. определение шага вдоль этого направления.

Методы построения таких последовательностей часто называют методами спуска, так как осуществляется переход от больших значений функций к меньшим. Математически методы спуска описываются соотношением $x_{k+1} = x_k + a_k \cdot p_k, k = 0, 1, 2, \ldots$, где p_k - вектор, определяющий направление спуска; a_k - длина шага.

Различные методы спуска отличаются друг от друга способами выбора двух параметров - направления спуска и длины шага вдоль этого направления. На практике применяются только методы, обладающие сходимостью. Они позволяют за конечное число шагов получить точку минимума или подойти к точке, достаточно близкой к точке минимума.

В методах спуска решение задачи теоретически получается за бесконечное число итераций. На практике вычисления прекращаются при выполнении некоторых критериев (условий) останова итерационного процесса.

Методы поиска точки минимума называются детерминированными, если оба элемента перехода от x_k к x_{k+1} (направление движения и величина шага) выбираются однозначно по доступной в точке x_k информации. Если же при переходе используется какой-либо случайный механизм, то алгоритм поиска называется случайным поиском минимума.

Детерминированные алгоритмы безусловной минимизации делят на классы в зависимости от вида используемой информации. Если на каждой итерации используются лишь значения минимизируемых функций, то метод называется методом нулевого порядка. Если, кроме того, требуется вычисление первых производных минимизируемой функции, то имеют место методы первого порядка, при необходимости дополнительного вычисления вторых производных - методы второго порядка.

Минимизация функций многих переменных.

Градиентом дифференцируемой функции $f(x)$ в точке x_0 называется n-мерный вектор $f(x_0)$, компоненты которого являются частными производными функции $f(x)$, вычисленными в точке x_0, т. е.

$$f'(x_0) = (\frac{df(x_0)}{dx_1}, \ldots, \frac{df(x_0)}{dx_n})^T.$$

Этот вектор перпендикулярен к плоскости, проведенной через точку x_0, и касательной к поверхности уровня функции $f(x)$, проходящей через точку x_0.В каждой точке такой поверхности функция $f(x)$ принимает одинаковое значение. Вектор-градиент направлен в сторону наискорейшего возрастания функции в данной точке. Вектор, противоположный градиенту ($-f'(x_0)$), называется антиградиентом и направлен в сторону наискорейшего убывания функции. В точке минимума градиент функции равен нулю. На свойствах градиента основаны методы первого порядка, называемые также градиентным и методами минимизации. Использование этих методов в общем случае позволяет определить точку локального минимума функции.

Очевидно, что если нет дополнительной информации, то из начальной точки x_0 разумно перейти в точку x_1, лежащую в направлении антиградиента - наискорейшего убывания функции. Выбирая в качестве направления спуска p_k антиградиент $-f'(x_k)$ в точке x_k, получаем итерационный процесс вида

$$x_{k+1} = x_k + a_k \cdot f'(x_k), a_k > 0, k = 0, 1, 2, \ldots$$

В качестве критерия останова итерационного процесса используют либо выполнение условия малости приращения аргумента

$$||x_k - x_{k+1}|| < \varepsilon,$$

либо выполнение условия малости градиента

$$||f'(x_{k+1})|| \le \gamma.$$

Здесь k - номер итерации; ε, γ - заданные величины точности решения задачи.Возможен и комбинированный критерий, состоящий в одновременном выполнении указанных условий. Градиентные методы отличаются друг от друга способами выбора величины шага a_k.

При методе с постоянным шагом для всех итераций выбирается некоторая постоянная величина шага. Достаточно малый шаг a_k обеспечит убывание функции, т. е. выполнение неравенства:

$$f(x_{k+1}) = f(x_k - akf'(x_k)) < f(x_k).$$

Однако это может привести к необходимости проводить неприемлемо большое количество итераций для достижения точки минимума. С другой стороны, слишком большой шаг может вызвать неожиданный рост функции либо привести к колебаниям около точки минимума (зацикливанию). Из-за сложности получения необходимой информации для выбора величины шага методы с постоянным шагом применяются на практике редко.

Более экономичны в смысле количества итераций и надежности градиентные методы с переменным шагом, когда в зависимости от результатов вычислений величина шага некоторым образом меняется из условия минимума функции $f(x)$ по a в направлении движения, т. е. в результате решения задачи одномерной минимизации:
$f(x_k + a_k \cdot p_k) = min(f(x_k + a \cdot p_k)), a > 0.$

Минимизация функций многих переменных.

Градиентом дифференцируемой функции $f(x)$ в точке x_0 называется n-мерный вектор $f(x_0)$, компоненты которого являются частными производными функции $f(x)$, вычисленными в точке x_0, т. е.

$$f'(x_0) = (\frac{df(x_0)}{dx_1}, \ldots, \frac{df(x_0)}{dx_n})^T.$$

Этот вектор перпендикулярен к плоскости, проведенной через точку x_0, и касательной к поверхности уровня функции $f(x)$, проходящей через точку x_0.В каждой точке такой поверхности функция $f(x)$ принимает одинаковое значение. Вектор-градиент направлен в сторону наискорейшего возрастания функции в данной точке. Вектор, противоположный градиенту $(-f'(x_0))$, называется антиградиентом и направлен в сторону наискорейшего убывания функции. В точке минимума градиент функции равен нулю. На свойствах градиента основаны методы первого порядка, называемые также градиентным и методами минимизации. Использование этих методов в общем случае позволяет определить точку локального минимума функции.

Очевидно, что если нет дополнительной информации, то из начальной точки x_0 разумно перейти в точку x_1, лежащую в направлении антиградиента - наискорейшего убывания функции. Выбирая в качестве направления спуска p_k антиградиент $-f'(x_k)$ в точке x_k, получаем итерационный процесс вида

$$x_{k+1} = x_k + a_k \cdot f'(x_k), a_k > 0, k = 0, 1, 2, \ldots$$

В качестве критерия останова итерационного процесса используют либо выполнение условия малости приращения аргумента

$$||x_k - x_{k+1}|| < \varepsilon,$$

либо выполнение условия малости градиента

$$||f'(x_{k+1})|| \leq \gamma.$$

Здесь k - номер итерации; ε, γ - заданные величины точности решения задачи.Возможен и комбинированный критерий, состоящий в одновременном выполнении указанных условий. Градиентные методы отличаются друг от друга способами выбора величины шага a_k.

При методе с постоянным шагом для всех итераций выбирается некоторая постоянная величина шага. Достаточно малый шаг a_k обеспечит убывание функции, т. е. выполнение неравенства:

$$f(x_{k+1}) = f(x_k - akf'(x_k)) < f(x_k).$$

Однако это может привести к необходимости проводить неприемлемо большое количество итераций для достижения точки минимума. С другой стороны, слишком большой шаг может вызвать неожиданный рост функции либо привести к колебаниям около точки минимума (зацикливанию). Из-за сложности получения необходимой информации для выбора величины шага методы с постоянным шагом применяются на практике редко.

Более экономичны в смысле количества итераций и надежности градиентные методы с переменным шагом, когда в зависимости от результатов вычислений величина шага некоторым образом меняется из условия минимума функции $f(x)$ по a в направлении движения, т. е. в результате решения задачи одномерной минимизации:
$f(x_k + a_k \cdot p_k) = min(f(x_k + a \cdot p_k)), a > 0.$

Практическая часть

Подключаем необходимые библиотеки

In [1]:

```
import numpy as np
import scipy.linalg as la
import random as r
```

Исследуемая функция

$$f(x, y) = \frac{(x-1)^2}{8} + \frac{(x-2)^2}{3}$$

In [2]:

```
def f(x):
    return (x[0] - 1)**2 / 8 + (x[1] - 2)**2 / 3

def df_1(x):
    return (x[0] - 1) / 4

def df_2(x):
    return 2  / 3  * (x[1] - 2)

def grad(x):
    return np.array([df_1(x) , df_2(x)])

x0, y0 = 454.95, -363.25
eps = 1e-10
```

Метод простого градиентного спуска

Алгоритм метода

1. Задаются начальное приближение и точность расчёта $\vec{x}^0, \quad \epsilon$
2. Рассчитывают $\vec{x}^{[j+1]} = \vec{x}^{[j]} - \lambda \nabla F(\vec{x}^{[j]})$, где $\lambda = const$
3. Проверяют условие остановки:
   - Если $|\vec{x}^{[j+1]} - \vec{x}^{[j]}| > \epsilon$, то $j = j + 1$ и переход к шагу 2.
   - Иначе $\vec{x} = \vec{x}^{[j+1]}$ и останов.

In [3]:

```
def meth1(x0, y0, print_flag):
    n = 1000
    x = np.zeros((n, 2))

    lamb_const = 1/2
    x[0] = x0, y0
    i = 0
    if print_flag:
        print(i, x[i])

    x[i + 1] = x[i] - lamb_const * grad(x[i])
    if print_flag:
        print(i + 1, x[i + 1])

    while la.norm(x[i + 1] - x[i]) > eps:
        i += 1
        x[i + 1] = x[i] - lamb_const * grad(x[i])
        if print_flag:
            print(i + 1, x[i + 1])

    i += 1
    if print_flag:
        print("Result:", x[i], "on step:", i)
    return x[i], i
```

Продемонстрируем работу алгоритма

In [4]:

```
meth1(x0, y0, True)
```

```
0 [ 454.95 -363.25]
1 [ 398.20625 -241.5    ]
2 [ 348.55546875 -160.33333333]
3 [ 305.11103516 -106.22222222]
4 [267.09715576 -70.14814815]
5 [233.83501129 -46.09876543]
6 [204.73063488 -30.06584362]
7 [179.26430552 -19.37722908]
8 [156.98126733 -12.25148605]
9 [137.48360891  -7.5009907 ]
10 [120.4231578  -4.3339938]
```

```
11 [105.49526307  -2.22266253]
12 [92.43335519 -0.81510836]
13 [81.00418579  0.1232611 ]
14 [71.00366257  0.74884073]
15 [62.25320475  1.16589382]
16 [54.59655415  1.44392921]
17 [47.89698488  1.62928614]
18 [42.03486177  1.75285743]
19 [36.90550405  1.83523829]
20 [32.41731605  1.89015886]
21 [28.49015154  1.92677257]
22 [25.0538826   1.95118171]
23 [22.04714727  1.96745448]
24 [19.41625386  1.97830298]
25 [17.11422213  1.98553532]
26 [15.09994436  1.99035688]
27 [13.33745132  1.99357125]
28 [11.7952699   1.99571417]
29 [10.44586117  1.99714278]
30 [9.26512852 1.99809519]
31 [8.23198746 1.99873012]
32 [7.32798902 1.99915342]
33 [6.5369904  1.99943561]
34 [5.8448666  1.99962374]
35 [5.23925827 1.99974916]
36 [4.70935099 1.99983277]
37 [4.24568211 1.99988852]
38 [3.83997185 1.99992568]
39 [3.48497537 1.99995045]
40 [3.17435345 1.99996697]
41 [2.90255927 1.99997798]
42 [2.66473936 1.99998532]
43 [2.45664694 1.99999021]
44 [2.27456607 1.99999348]
45 [2.11524531 1.99999565]
46 [1.97583965 1.9999971 ]
47 [1.85385969 1.99999807]
48 [1.74712723 1.99999871]
49 [1.65373633 1.99999914]
50 [1.57201929 1.99999943]
51 [1.50051688 1.99999962]
52 [1.43795227 1.99999975]
53 [1.38320823 1.99999983]
54 [1.3353072  1.99999989]
55 [1.2933938  1.99999992]
56 [1.25671958 1.99999995]
57 [1.22462963 1.99999997]
58 [1.19655093 1.99999998]
59 [1.17198206 1.99999999]
60 [1.1504843  1.99999999]
61 [1.13167377 1.99999999]
62 [1.11521454 2.        ]
63 [1.10081273 2.        ]
64 [1.08821114 2.        ]
65 [1.07718474 2.        ]
66 [1.06753665 2.        ]
67 [1.05909457 2.        ]
68 [1.05170775 2.        ]
69 [1.04524428 2.        ]
70 [1.03958874 2.        ]
71 [1.03464015 2.        ]
72 [1.03031013 2.        ]
73 [1.02652137 2.        ]
74 [1.0232062 2.       ]
75 [1.02030542 2.        ]
76 [1.01776724 2.        ]
77 [1.01554634 2.        ]
78 [1.01360305 2.        ]
79 [1.01190266 2.        ]
80 [1.01041483 2.        ]
81 [1.00911298 2.        ]
82 [1.00797386 2.        ]
83 [1.00697712 2.        ]
84 [1.00610498 2.        ]
85 [1.00534186 2.        ]
86 [1.00467413 2.        ]
87 [1.00408986 2.        ]
88 [1.00357863 2.        ]
89 [1.0031313 2.       ]
90 [1.00273989 2.        ]
```

```
91 [1.0023974 2.       ]
92 [1.00209773 2.        ]
93 [1.00183551 2.        ]
94 [1.00160607 2.        ]
95 [1.00140531 2.        ]
96 [1.00122965 2.        ]
97 [1.00107594 2.        ]
98 [1.00094145 2.        ]
99 [1.00082377 2.        ]
100 [1.0007208 2.       ]
101 [1.0006307 2.       ]
102 [1.00055186 2.        ]
103 [1.00048288 2.        ]
104 [1.00042252 2.        ]
105 [1.0003697 2.       ]
106 [1.00032349 2.        ]
107 [1.00028305 2.        ]
108 [1.00024767 2.        ]
109 [1.00021671 2.        ]
110 [1.00018962 2.        ]
111 [1.00016592 2.        ]
112 [1.00014518 2.        ]
113 [1.00012703 2.        ]
114 [1.00011115 2.        ]
115 [1.00009726 2.        ]
116 [1.0000851 2.       ]
117 [1.00007446 2.        ]
118 [1.00006516 2.        ]
119 [1.00005701 2.        ]
120 [1.00004989 2.        ]
121 [1.00004365 2.        ]
122 [1.00003819 2.        ]
123 [1.00003342 2.        ]
124 [1.00002924 2.        ]
125 [1.00002559 2.        ]
126 [1.00002239 2.        ]
127 [1.00001959 2.        ]
128 [1.00001714 2.        ]
129 [1.000015 2.      ]
130 [1.00001312 2.        ]
131 [1.00001148 2.        ]
132 [1.00001005 2.        ]
133 [1.00000879 2.        ]
134 [1.00000769 2.        ]
135 [1.00000673 2.        ]
136 [1.00000589 2.        ]
137 [1.00000515 2.        ]
138 [1.00000451 2.        ]
139 [1.00000395 2.        ]
140 [1.00000345 2.        ]
141 [1.00000302 2.        ]
142 [1.00000264 2.        ]
143 [1.00000231 2.        ]
144 [1.00000202 2.        ]
145 [1.00000177 2.        ]
146 [1.00000155 2.        ]
147 [1.00000136 2.        ]
148 [1.00000119 2.        ]
149 [1.00000104 2.        ]
150 [1.00000091 2.        ]
151 [1.00000079 2.        ]
152 [1.0000007 2.       ]
153 [1.00000061 2.        ]
154 [1.00000053 2.        ]
155 [1.00000047 2.        ]
156 [1.00000041 2.        ]
157 [1.00000036 2.        ]
158 [1.00000031 2.        ]
159 [1.00000027 2.        ]
160 [1.00000024 2.        ]
161 [1.00000021 2.        ]
162 [1.00000018 2.        ]
163 [1.00000016 2.        ]
164 [1.00000014 2.        ]
165 [1.00000012 2.        ]
166 [1.00000011 2.        ]
167 [1.00000009 2.        ]
168 [1.00000008 2.        ]
169 [1.00000007 2.        ]
170 [1.00000006 2.        ]
```

```
171 [1.00000006 2.        ]
172 [1.00000005 2.        ]
173 [1.00000004 2.        ]
174 [1.00000004 2.        ]
175 [1.00000003 2.        ]
176 [1.00000003 2.        ]
177 [1.00000002 2.        ]
178 [1.00000002 2.        ]
179 [1.00000002 2.        ]
180 [1.00000002 2.        ]
181 [1.00000001 2.        ]
182 [1.00000001 2.        ]
183 [1.00000001 2.        ]
184 [1.00000001 2.        ]
185 [1.00000001 2.        ]
186 [1.00000001 2.        ]
187 [1.00000001 2.        ]
188 [1.00000001 2.        ]
189 [1. 2.]
190 [1. 2.]
191 [1. 2.]
192 [1. 2.]
193 [1. 2.]
194 [1. 2.]
195 [1. 2.]
196 [1. 2.]
197 [1. 2.]
198 [1. 2.]
199 [1. 2.]
200 [1. 2.]
201 [1. 2.]
202 [1. 2.]
203 [1. 2.]
204 [1. 2.]
Result: [1. 2.] on step: 204
```

```
Out[4]:
```

```
(array([1., 2.]), 204)
```

Метод наискорейшего спуска

Алгоритм метода

1. Задаются начальное приближение и точность расчёта $\vec{x}^0, \quad \epsilon$
2. Рассчитывают $\vec{x}^{[j+1]} = \vec{x}^{[j]} - \lambda^{[j]} \nabla F(\vec{x}^{[j]})$, где $\lambda^{[j]} = \text{argmin}_\lambda F(\vec{x}^{[j]} - \lambda^{[j]} \nabla F(\vec{x}^{[j]}))$
3. Проверяют условие остановки:
   - Если $|\vec{x}^{[j+1]} - \vec{x}^{[j]}| > \epsilon$, то $j = j + 1$ и переход к шагу 2.
   - Иначе $\vec{x} = \vec{x}^{[j+1]}$ и останов.

In [5]:

```
def meth2(x0, y0, print_flag):
    n = 1000
    x = np.zeros((n, 2))

    lamb = np.zeros(n)
    def cal_lamb(x, i):
        a = grad(x[i])[0]
        b = grad(x[i])[1]
        xx = x[i][0]
        yy = x[i][1]
        return (a**2 + b**2) / (0.25 * a**2 + b**2 *2 / 3)

    x[0] = x0, y0
    i = 0
    if print_flag:
        print(i, x[i])

    lamb[i] = cal_lamb(x, i)
    x[i + 1] = x[i] - lamb[i] * grad(x[i])
    if print_flag:
        print(i + 1, x[i + 1])

    while la.norm(x[i + 1] - x[i]) > eps:
        i += 1
        lamb[i] = cal_lamb(x, i)
        x[i + 1] = x[i] - lamb[i] * grad(x[i])
        if print_flag:
            print(i + 1, x[i + 1])

    i += 1
    if print_flag:
        print("Result:", x[i], "on step:", i)
    return x[i], i
```

Продемонстрируем работу алгоритма

In [6]:

```
meth2(x0, y0, True)
```

```
0 [ 454.95 -363.25]
1 [263.34857383  47.85207973]
2 [ 61.14151986 -46.39010933]
3 [35.75722428  8.07470815]
4 [ 8.96784318 -4.41095874]
5 [5.60480735 2.8048071 ]
6 [2.0556189  1.15064478]
7 [1.61006744 2.10662479]
8 [1.13985356 1.88747326]
9 [1.08082472 2.01412617]
10 [1.01852849 1.98509191]
11 [1.01070806 2.0018715 ]
12 [1.00245474 1.9980249 ]
13 [1.00141866 2.00024795]
14 [1.00032522 1.99973833]
15 [1.00018795 2.00003285]
16 [1.00004309 1.99996533]
17 [1.0000249  2.00000435]
18 [1.00000571 1.99999541]
19 [1.0000033  2.00000058]
20 [1.00000076 1.99999939]
21 [1.00000044 2.00000008]
22 [1.0000001  1.99999992]
23 [1.00000006 2.00000001]
24 [1.00000001 1.99999999]
25 [1.00000001 2.        ]
26 [1. 2.]
27 [1. 2.]
28 [1. 2.]
29 [1. 2.]
30 [1. 2.]
31 [1. 2.]
Result: [1. 2.] on step: 31
```

Out[6]:

```
(array([1., 2.]), 31)
```

Метод сопряженных градиентов

Алгоритм метода

1. Задаются начальным приближением и погрешностью: $\vec{x}_0, \quad \varepsilon, \quad k = 0 <$
2. Рассчитывают начальное направление: $j = 0, \quad \vec{S}_k^j = -\nabla f(\vec{x}_k), \quad \vec{x}_k^j = \vec{x}_k$
3. Рассчитывают $\vec{x}_k^{j+1} = \vec{x}_k^j + \lambda \vec{S}_k^j, \quad \lambda = \arg\min_\lambda f(\vec{x}_k^j + \lambda \vec{S}_k^j), \quad \vec{S}_k^{j+1} = -\nabla f(\vec{x}_k^{j+1}) + \omega \vec{S}_k^j, \omega = \frac{||\nabla f(\vec{x}_k^{j+1})||^2}{||\nabla f(\vec{x}_k^j)||^2}$
   - Если $||\vec{S}_k^{j+1}|| < \varepsilon$ или $||\vec{x}_k^{j+1} - \vec{x}_k^j|| < \varepsilon$, то $\vec{x} = \vec{x}_k^{j+1}$ и останов.
   - Иначе если $(j+1)

In [7]:

```
def meth3(x0, y0, print_flag):
    n = 1000
    x = np.zeros((n, 2))


    lamb = np.zeros(n)
    def cal_lamb(x, i):
        a = grad(x[i])[0]
        b = grad(x[i])[1]
        xx = x[i][0]
        yy = x[i][1]
        return (a**2 + b**2) / (0.25 * a**2 + b**2 *2 / 3)

    x[0] = x0, y0
    i = 0
    k = 0
    if print_flag:
        print(i, x[i])

    s = grad(x[k])
    lamb[i] = cal_lamb(x, i)
    x[i + 1] = x[i] + lamb[i] * s[i]
    omega = la.norm(grad(x[i + 1])) ** 2 / la.norm(grad(x[i])) **2
    s *= omega
    s += -grad(x[i + 1])
    if (print_flag):
        print(i + 1, x[i + 1])

    while la.norm(x[i + 1] - x[i]) > eps:
        i += 1
        lamb[i] = cal_lamb(x, i)
        x[i + 1] = x[i] + lamb[i] * s
        omega = la.norm(grad(x[i + 1])) ** 2 / la.norm(grad(x[i])) **2
        s *= omega
        s += -grad(x[i + 1])
        if (print_flag):
            print(i + 1, x[i + 1])

    i += 1
    if (print_flag):
        print("Result:", x[i], "on step:", i)
    return x[i], i
```

Продемонстрируем работу алгоритма

In [8]:

```
meth3(x0, y0, True)
```

```
0 [ 454.95 -363.25]
1 [ 646.55142617 -171.64857383]
2 [ 392.80033855 -215.90037501]
3 [66.07790486 29.7639999 ]
4 [25.38810183 -3.02013781]
5 [3.25328776 2.98330676]
6 [1.86102671 1.71019431]
7 [1.1920549  1.99325691]
8 [0.97078237 2.02450482]
9 [0.97007326 1.99897897]
10 [0.99930153 1.99106269]
11 [1.00656193 1.99810934]
12 [1.00394804 2.00262171]
13 [1.00044442 2.00257076]
14 [0.99809428 1.99996151]
15 [0.99951076 1.99953062]
16 [0.99999935 1.99995461]
17 [1.00000323 2.00000316]
18 [1.00000194 2.00000002]
19 [0.99999986 1.99999962]
20 [0.99999969 1.99999996]
21 [0.99999992 2.00000013]
22 [1.00000007 2.00000008]
23 [1.00000011 1.99999997]
24 [1.00000006 1.99999996]
25 [1. 2.]
26 [1. 2.]
27 [1. 2.]
28 [1. 2.]
29 [1. 2.]
Result: [1. 2.] on step: 29
```

Out[8]:

```
(array([1., 2.]), 29)
```

Сравнение алгоритмов

Так как алгоритмы могут давать существенно разные результаты для различных точек, то будет суммировать из результаты, полученные на большом числе итераций, для каждой из которых будем генерировать случайную точку.

In [9]:

```
n = 5000
s = np.zeros((3, n))

for i in range(0, n):
    x, y = r.random(), r.random()
    s[0][i] = meth1(x, y, False)[1]
    s[1][i] = meth2(x, y, False)[1]
    s[2][i] = meth3(x, y, False)[1]

print("Статистика за {} итераций для достижения точности в 1e-10".format(n))
print("1.Метод наискорейшего спуска")
print("2.Метод сопряженных градиентов")
print("3.Метод простого градиентного спуска")
print()

print("Суммарное число шагов")
print(np.sum(s[0]), np.sum(s[1]), np.sum(s[2]))
print()

print("Среднее число шагов")
print(np.average(s[0]), np.average(s[1]), np.average(s[2]))
print()

print("Максимальное число шагов")
print(np.max(s[0]), np.max(s[1]), np.max(s[2]))
print()

print("Минимальное число шагов")
print(np.min(s[0]), np.min(s[1]), np.min(s[2]))
print()
```

```
Статистика за 5000 итераций для достижения точности в 1e-10
1.Метод наискорейшего спуска
2.Метод сопряженных градиентов
3.Метод простого градиентного спуска

Суммарное число шагов
755245.0 64241.0 135467.0

Среднее число шагов
151.049 12.8482 27.0934

Максимальное число шагов
158.0 22.0 47.0

Минимальное число шагов
96.0 4.0 18.0
```

Вывод практической части

По статистике, собранной нами при сравнении алгоритмов, можно сказать, что метод наискорейшего спуска является самым оптимальным для нашей функции.

При различных оптимизируемых функциях и начальных точках, результаты (среднее, максимальное и минимальное число шагов) могут сильно отличаться.

Полный список методов от лучшего к худшему по эффективности:

1. Метод наискорейшего спуска
2. Метод сопряженных градиентов
3. Метод простого градиентного спуска