Лабораторная 2

Моисеев M32001, Муров M32011

Вариант 3

Траектории

Функция: $f(x, y) = 10x^2 + y^2$

Начальное приближение: $(10, 10)$

Искомая точность: 10^{-5} в значении

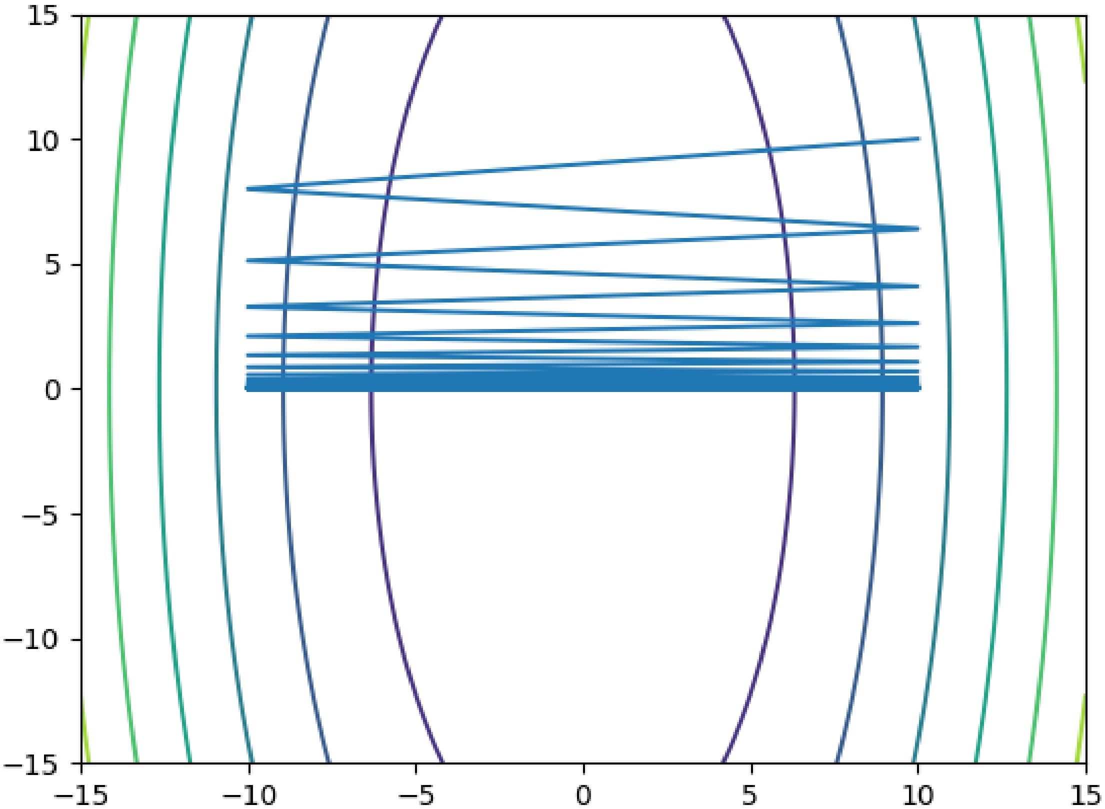

Figure 1: Постоянная 0.1. 35 итераций

Начальная точка: [10, 10]

название метода	10x$\hat{2}$+y$\hat{2}$	10000x$\hat{2}$+10000y$\hat{2}$	100000x$\hat{2}$+0.00001y$\hat{2}$
brent	9	2	2
break h0=0.7 eps=0.1 lambda=0.95	31	16	16
break h0=0.5 eps=0.9 lambda=0.9	50	60	64
golden	9	2	2
fibonacci	9	3	4

Начальная точка: [1, 100]

название метода	10x$\hat{2}$+y$\hat{2}$	10000x$\hat{2}$+10000y$\hat{2}$	100000x$\hat{2}$+0.00001y$\hat{2}$
brent	9	2	1932
break h0=0.7 eps=0.1 lambda=0.95	36	18	14
break h0=0.5 eps=0.9 lambda=0.9	50	69	53
golden	9	2	2
fibonacci	9	3	4

Начальная точка: [1, 10000]

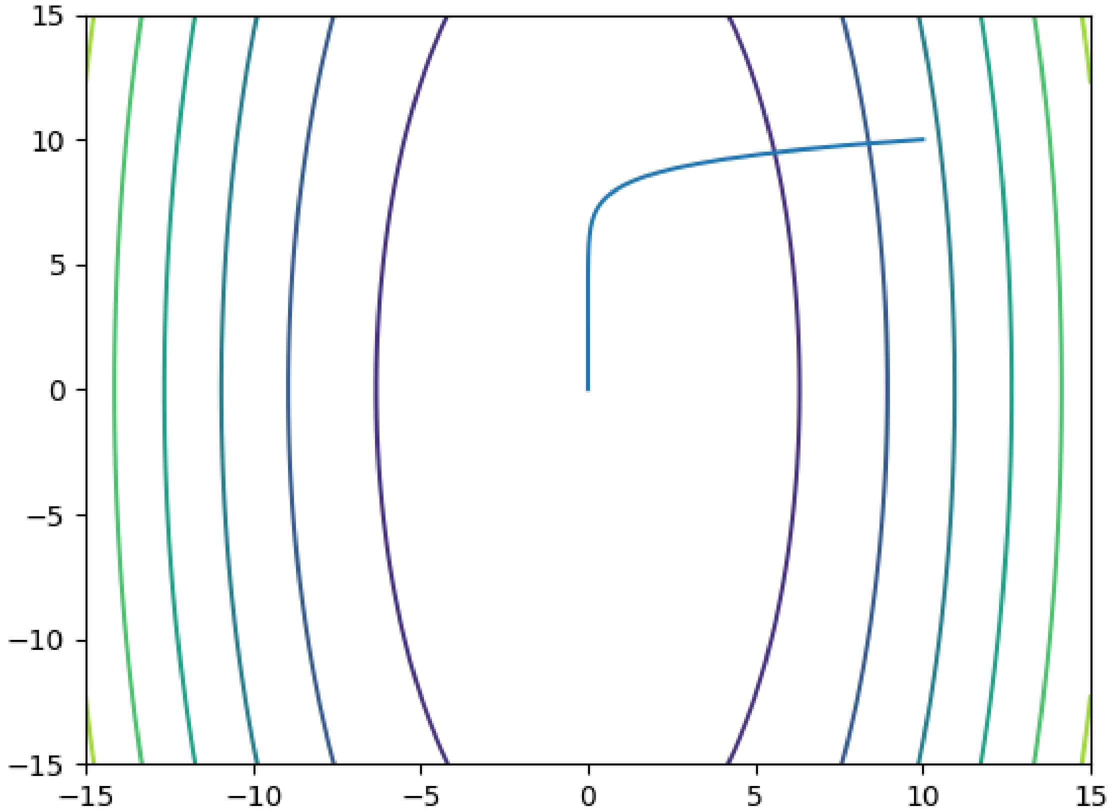

Figure 2: Постоянная 0.01. 320 итераций

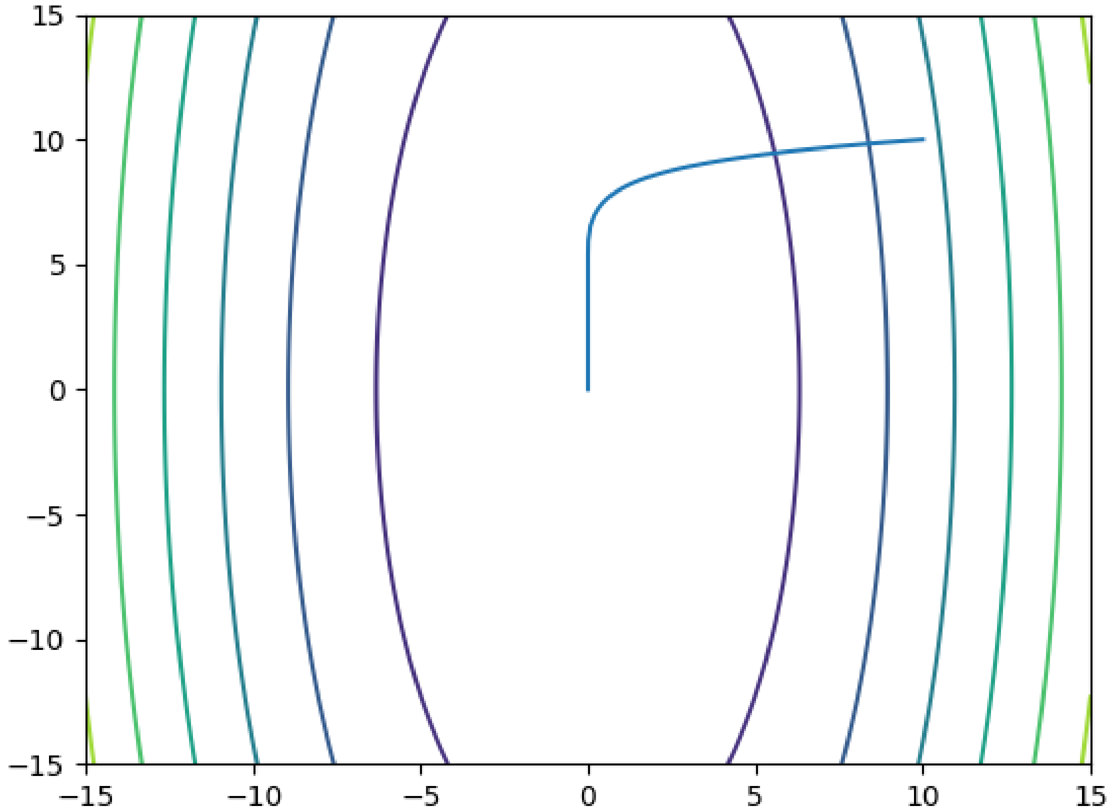

Figure 3: Дробление 1 0.95 0.95. 95 итераций

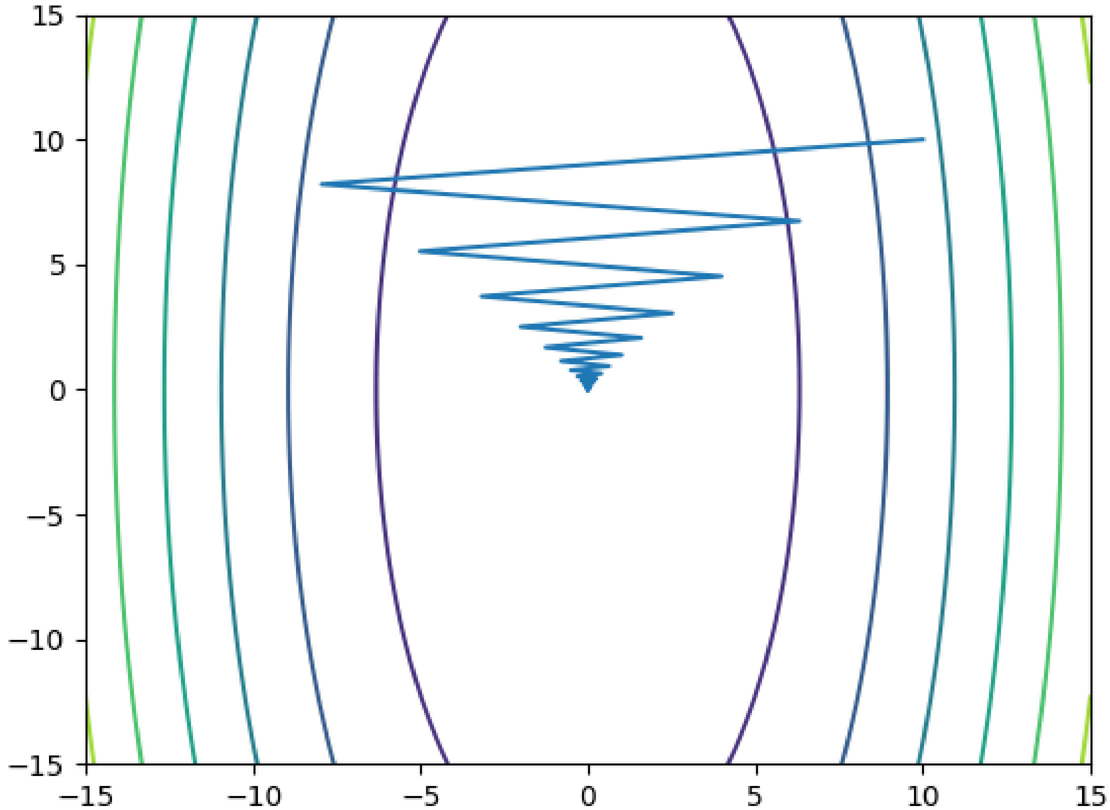

Figure 4: Дробление 1 0.1 0.95. 42 итерации

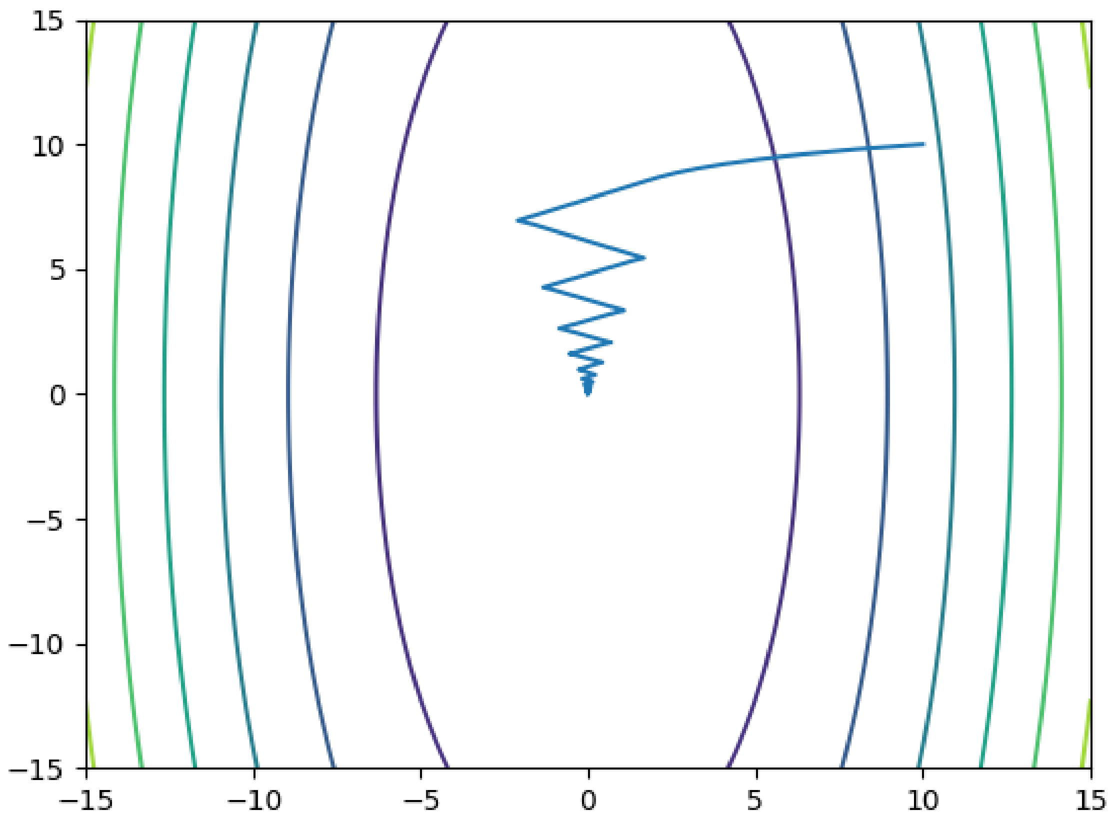

Figure 5: Дробление 1 0.1 0.1. 72 итерации

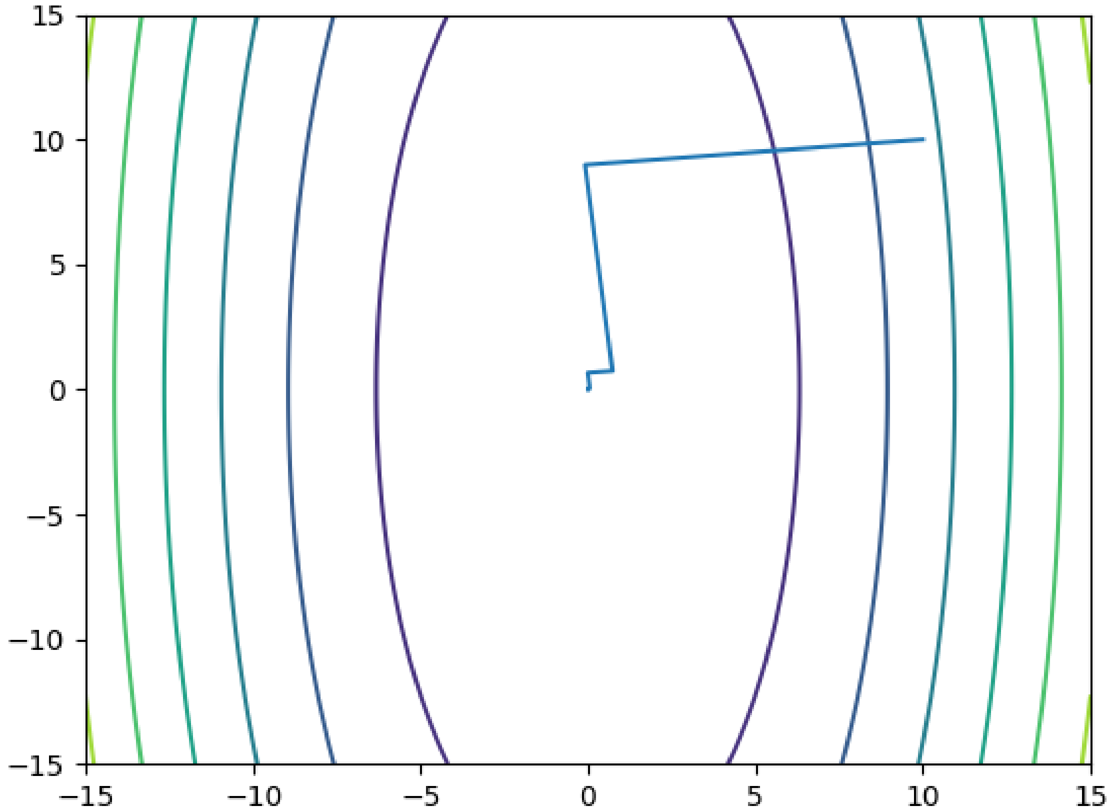

Figure 6: Золотое сечение. 9 итераций

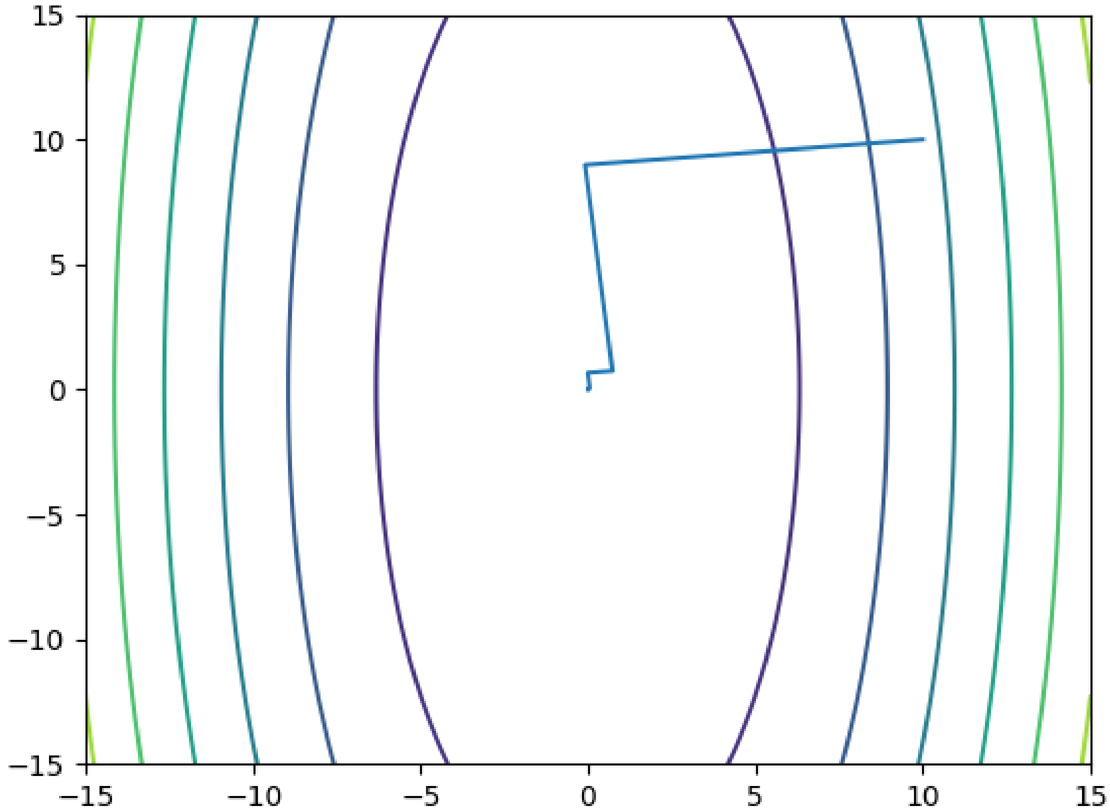

Figure 7: Фибоначчи. 9 итераций

название метода	$10x\hat{2}+y\hat{2}$	$10000x\hat{2}+10000y\hat{2}$	$100000x\hat{2}+0.00001y\hat{2}$
brent	4	2	11
break h0=0.7 eps=0.1 lambda=0.95	47	23	14
break h0=0.5 eps=0.9 lambda=0.9	72	90	53
golden	4	2	776
fibonacci	4	4	4

Начальная точка: [100, 1]

название метода	$10x\hat{2}+y\hat{2}$	$10000x\hat{2}+10000y\hat{2}$	$100000x\hat{2}+0.00001y\hat{2}$
brent	3	2	2
break h0=0.7 eps=0.1 lambda=0.95	25	18	19
break h0=0.5 eps=0.9 lambda=0.9	61	69	75
golden	3	2	2
fibonacci	4	3	4

Начальная точка: [10000, 1]

название метода	$10x\hat{2}+y\hat{2}$	$10000x\hat{2}+10000y\hat{2}$	$100000x\hat{2}+0.00001y\hat{2}$
brent	3	2	2
break h0=0.7 eps=0.1 lambda=0.95	27	23	24
break h0=0.5 eps=0.9 lambda=0.9	84	90	97
golden	3	2	2
fibonacci	5	4	5

Начальная точка: [1000, 10000]

название метода	$10x\hat{2}+y\hat{2}$	$10000x\hat{2}+10000y\hat{2}$	$100000x\hat{2}+0.00001y\hat{2}$
brent	74	2	439
break h0=0.7 eps=0.1 lambda=0.95	51	23	22
break h0=0.5 eps=0.9 lambda=0.9	75	90	86
golden	74	2	2
fibonacci	74	4	5

Сравнение метода сопряженных градиентов с обычным градиентным спуском Тесты с хорошим числом обусловленности:

Для небольших чисел обусловленности намного лучше себя показывает обычный градиентный спуск Тесты с плохим числом обусловленности:

При больших числах обусловленности, метод сопряженных градиентов стабилизируется и видно, что его эффективность не зависит от него.

```
Method:brent
size: 2
k=10.1 steps=9
k=1000.0010000000001 steps=4
k=100000000.0 steps=4
size: 3
k=2.9999999999999996 steps=2
k=10001.0001 steps=9053
k=10000499.98800075 steps=7
size: 4
k=4.0 steps=2
k=173.23394673100304 steps=7
k=17320.508364363905 steps=5
k=141774.46951415477 steps=13770
size: 6
```

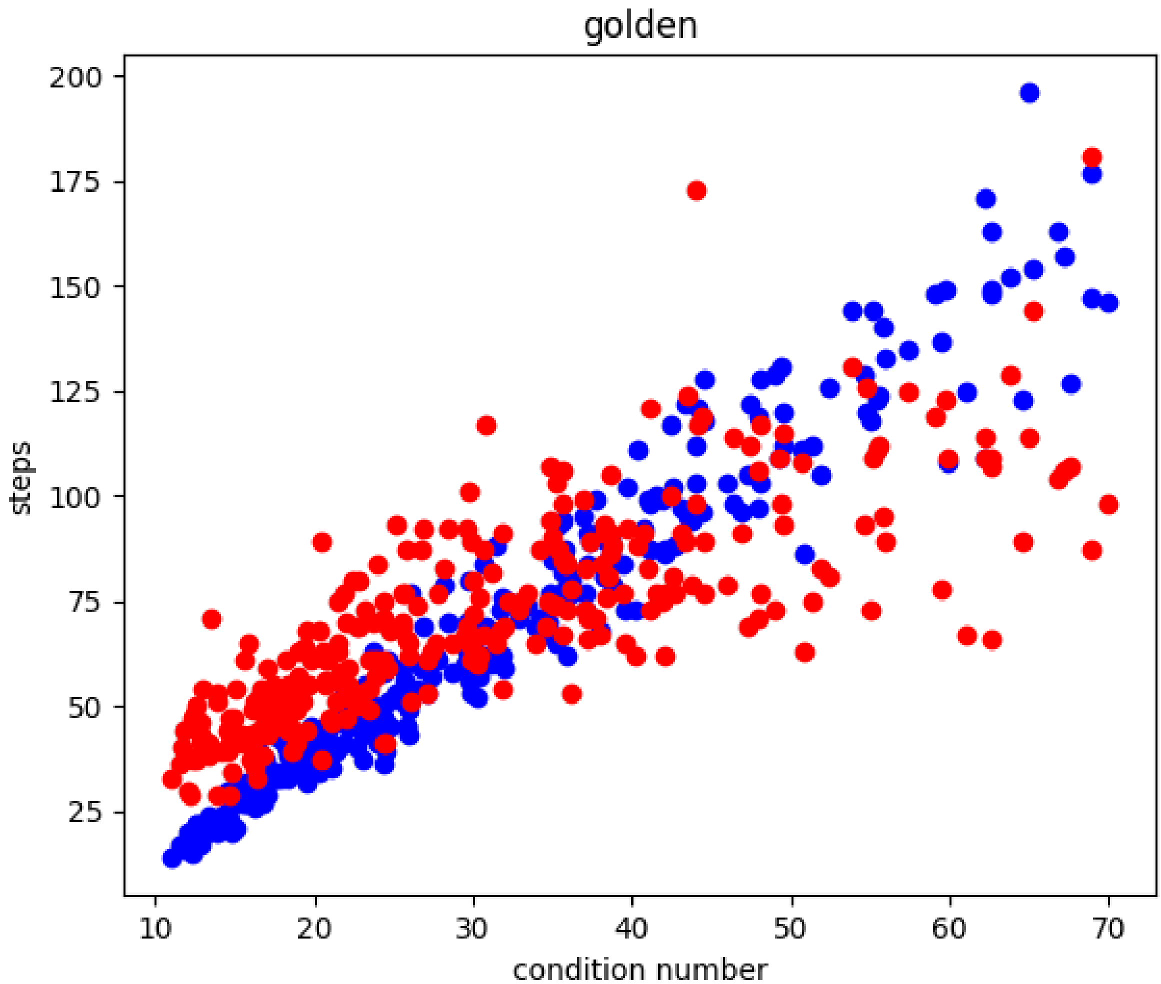

Figure 8: Размер задачи 10:

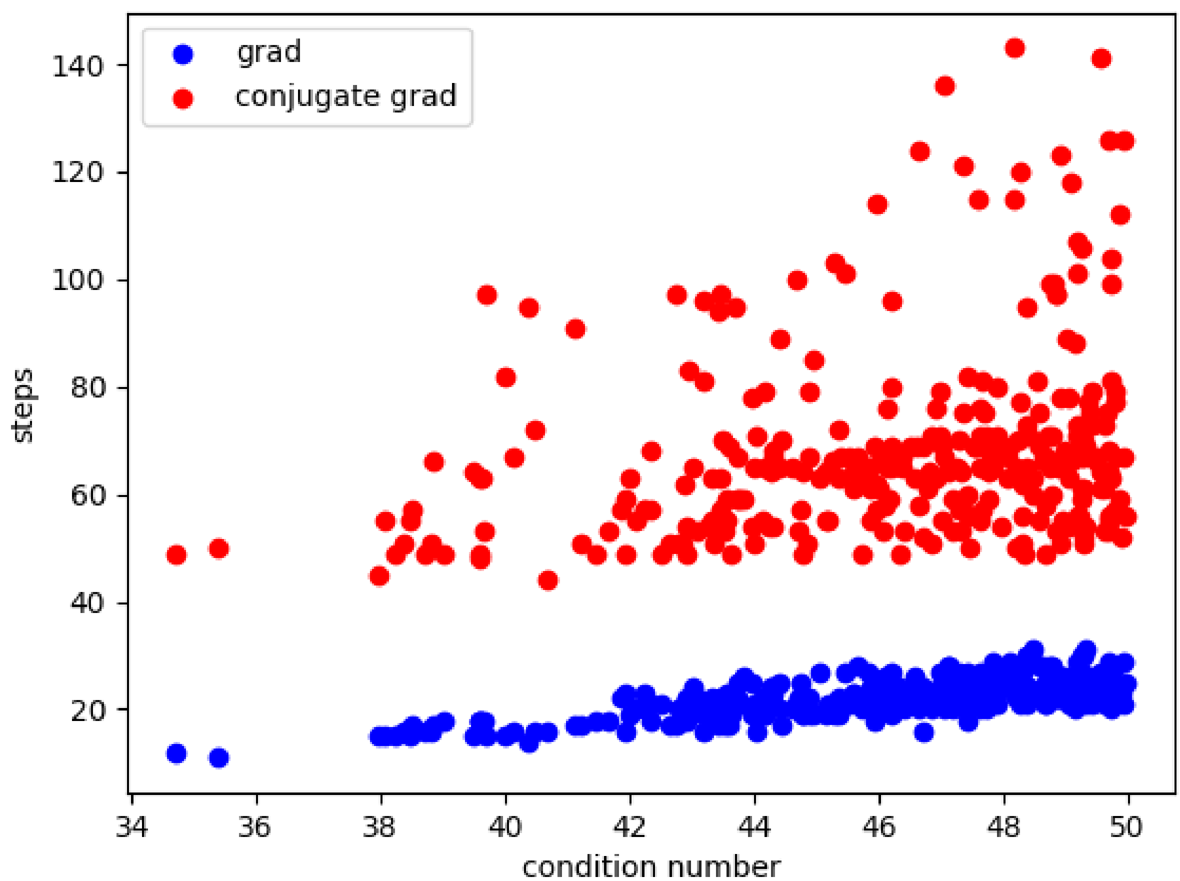

Figure 9: Размер задачи 30:

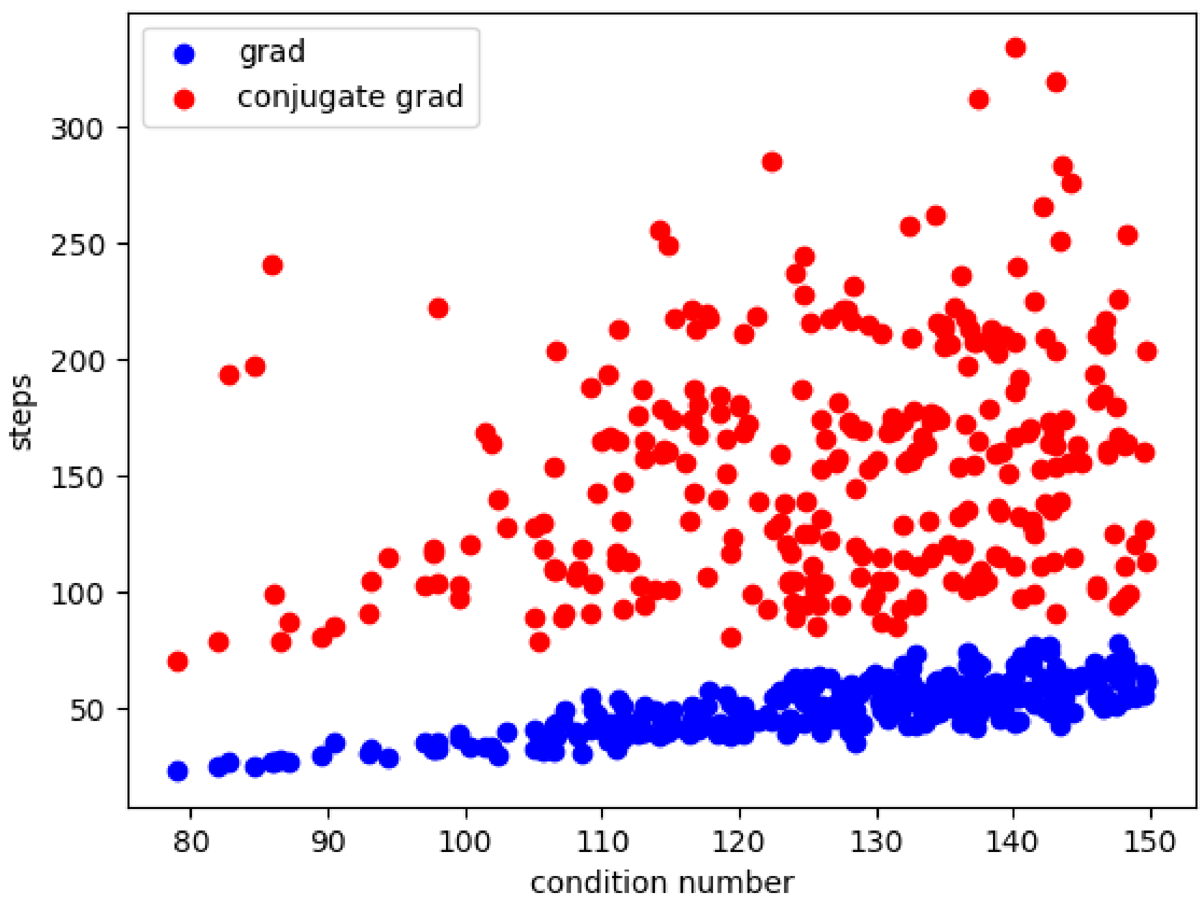

Figure 10: Размер задачи 50:

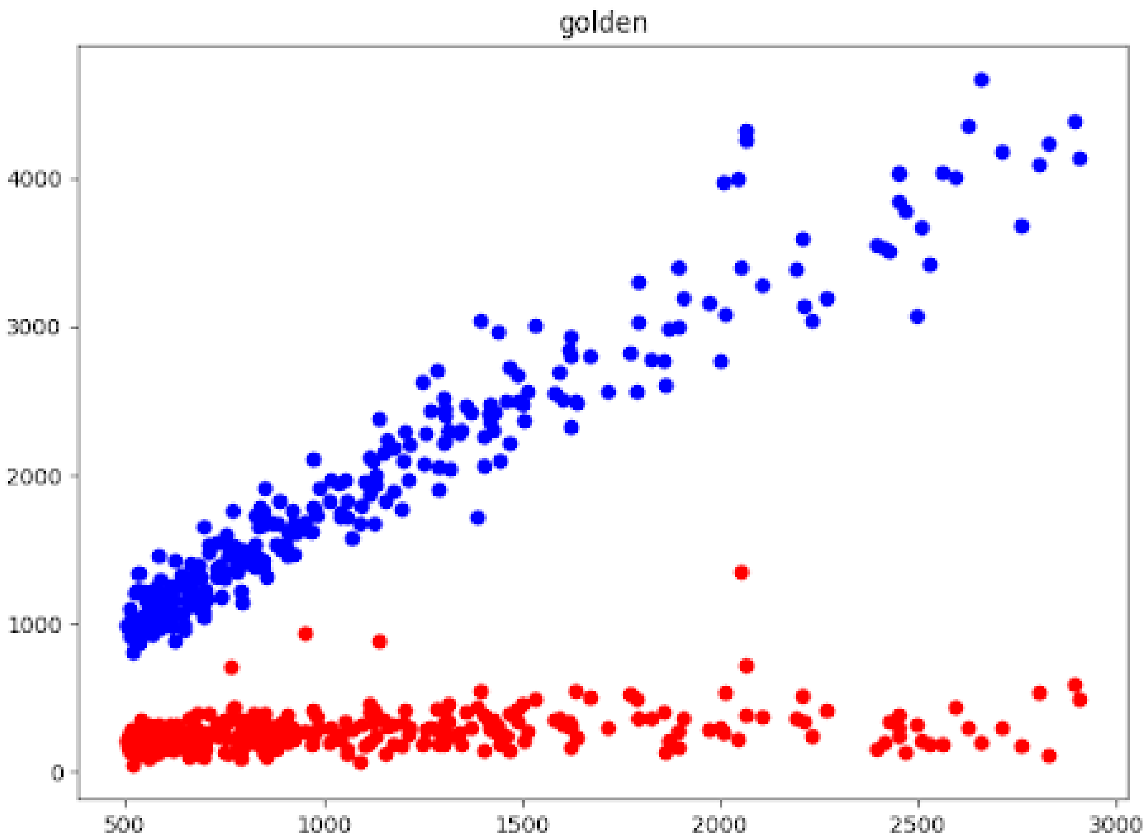

Figure 11: Размер задачи 3:

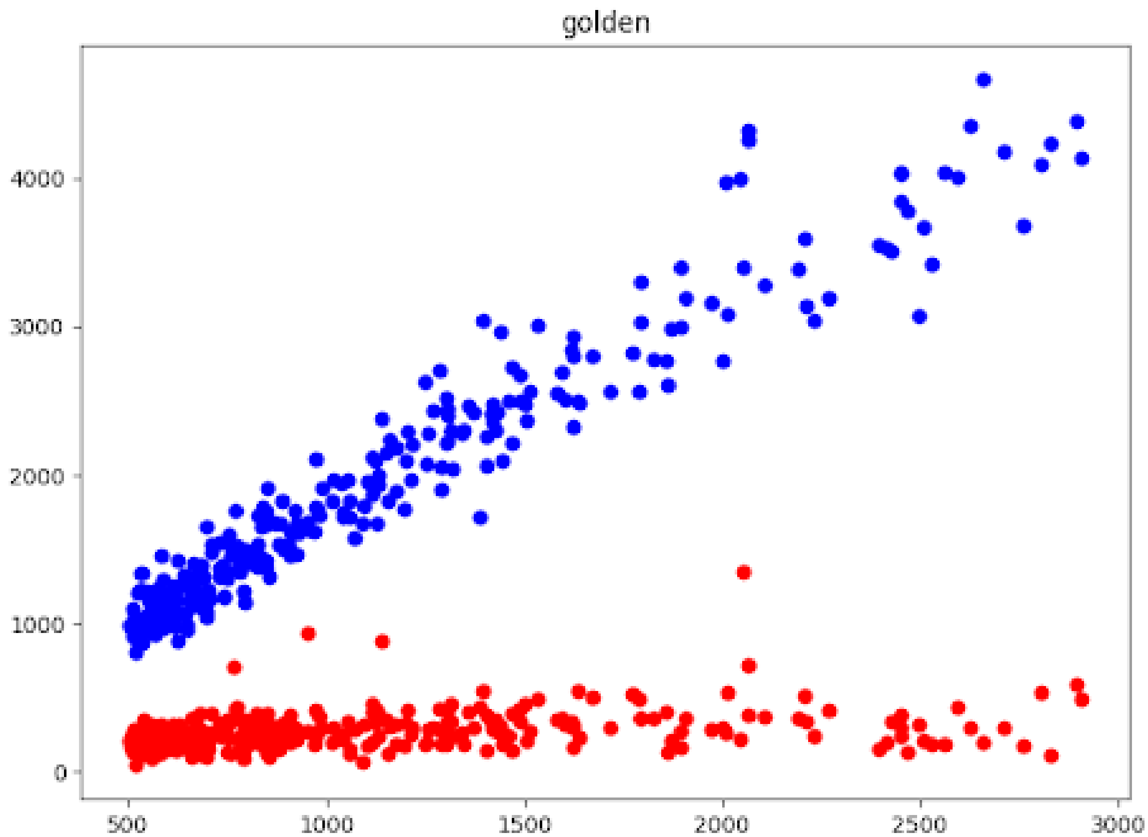

Figure 12: Размер задачи 10:

```
k=7.4301405355945205 steps=13
k=2012487.2896767037 steps=2059
k=18.55397531527947 steps=40
Method:break h0=0.7 eps=0.1 lambda=0.95
size: 2
k=10.1 steps=31
k=1000.0010000000001 steps=857
k=100000000.0 steps=15
size: 3
k=2.9999999999999996 steps=11
k=10001.0001 steps=8485
k=10000499.98800075 steps=211
size: 4
k=4.0 steps=11
k=173.23394673100304 steps=150
k=17320.508364363905 steps=9640
k=141774.46951415477 steps=67565
size: 6
k=7.4301405355945205 steps=17
k=2012487.2896767037 steps=9574
k=18.55397531527947 steps=38
Method:break h0=0.5 eps=0.9 lambda=0.9
size: 2
k=10.1 steps=50
k=1000.0010000000001 steps=888
k=100000000.0 steps=58
size: 3
k=2.9999999999999996 steps=41
k=10001.0001 steps=3631
k=10000499.98800075 steps=234
size: 4
k=4.0 steps=42
k=173.23394673100304 steps=211
k=17320.508364363905 steps=4035
k=141774.46951415477 steps=28588
size: 6
k=7.4301405355945205 steps=47
k=2012487.2896767037 steps=846
k=18.55397531527947 steps=50
Method:golden
size: 2
k=10.1 steps=9
k=1000.0010000000001 steps=4
k=100000000.0 steps=2
size: 3
k=2.9999999999999996 steps=2
k=10001.0001 steps=9087
k=10000499.98800075 steps=7
size: 4
k=4.0 steps=2
k=173.23394673100304 steps=7
k=17320.508364363905 steps=5
k=141774.46951415477 steps=24225
```

```
size: 6
k=7.4301405355945205 steps=13
k=2012487.2896767037 steps=2058
k=18.55397531527947 steps=40
Method:fibonacci
size: 2
k=10.1 steps=9
k=1000.0010000000001 steps=6
k=100000000.0 steps=3
size: 3
k=2.9999999999999996 steps=2
k=10001.0001 steps=58
k=10000499.98800075 steps=136
size: 4
k=4.0 steps=2
k=173.23394673100304 steps=7
k=17320.508364363905 steps=11621
k=141774.46951415477 steps=53095
size: 6
k=7.4301405355945205 steps=13
k=2012487.2896767037 steps=1808
k=18.55397531527947 steps=40
```

Код

```
from functools import partial, cache
from itertools import count

import numpy as np
from scipy.optimize import minimize_scalar, bracket, OptimizeResult
from toolz import comp


_epsilon = np.sqrt(np.finfo(float).eps)


@cache
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)


def norm_sq(a):
    return np.dot(a, a)


def next_grad(x, df, h):
    return x - h * df(x)


def const_h(c):
```

```
    def get_h(**kwargs):
        return c

    return get_h


def break_h(h0, eps, lam):
    def get_h(*, f, df, x, **kwargs):
        h = h0
        while not (f(x) - f(next_grad(x, df, h)) >= eps * h * norm_sq(df(x))):
            h *= lam
        return h

    return get_h


def fib_method(func, brack=None, args=(), xtol=_epsilon, **unknown):
    tol = xtol
    f = lambda x: func(*(x,) + args)
    if brack is None:
        xa, xb, xc, fa, fb, fc, funcalls = bracket(func, args=args)
    elif len(brack) == 2:
        xa, xb, xc, fa, fb, fc, funcalls = bracket(func, *brack, args=args)
    elif len(brack) == 3:
        xa, xb, xc = brack
        xa, xc = sorted([xa, xc])
        if not (xa < xb < xc):
            raise ValueError("Not a bracketing interval.")
        fa, fb, fc = map(f, [xa, xb, xc])
        if not (fb < fa and fb < fc):
            raise ValueError("Not a bracketing interval.")
        funcalls = 3
    else:
        raise ValueError("Not a bracketing interval.")

    xa, xc = sorted([xa, xc])
    n = next(i for i in count() if fibonacci(i) * tol > xc - xa)
    w = (xc - xa) * fibonacci(n) / fibonacci(n + 2)
    x1, x2 = xa + w, xc - w
    for i in range(n):
        if f(x1) < f(x2):
            x2, xc = x1, x2
            x1 = xa + xc - x2
        else:
            xa, x1 = x1, x2
            x2 = xa + xc - x1
    res = (xa + xc) / 2
    funcalls += n
    return OptimizeResult(
        fun=f(res),
        nfev=funcalls,
        x=res,
        nit=n,
```

```python
        success=True,
        message="Success",
    )


def steepest_h(method):
    def get_h(*, f, df, x, **kwargs):
        f1d = comp(f, partial(next_grad, x, df))
        return minimize_scalar(f1d, method=method).x

    return get_h


def stop_x(e):
    def stop(*, x, px, **kwargs):
        return norm_sq(x - px) <= e**2

    return stop


def stop_f(e):
    def stop(*, x, px, f, **kwargs):
        return abs(f(x) - f(px)) <= e

    return stop


def grad(f, df, x, get_h, stop, maxi=None):
    yield x
    px = x
    for i in count():
        if i == maxi:
            return
        x = next_grad(x, df, get_h(f=f, df=df, x=x))
        if stop(x=x, px=px, f=f):
            return
        yield x
        px = x


__all__ = (
    "next_grad",
    "const_h",
    "break_h",
    "steepest_h",
    "stop_x",
    "stop_f",
    "grad",
    "norm_sq",
    "fib_method",
)
```

```python
import numpy as np
```

```python
from scipy.optimize import minimize_scalar

import primat.lab2.grad


def extra_grad(f, df, x, method, stop, maxi=None):
    yield x
    n = x.shape[0]
    i = 0
    while 1:
        s = df(x)
        for j in range(n):
            i += 1
            if i == maxi:
                return
            lam = minimize_scalar(lambda lam: f(x - lam * s), method=method).x
            px = x
            x = x - lam * s
            omega = grad.norm_sq(df(x)) / grad.norm_sq(df(px))
            # omega = max(0, np.dot(df(x), df(x) - df(px)) / norm_sq(df(px)))
            s = df(x) - omega * s
            yield x
            if stop(x=x, px=px, f=f, s=s):
                return


__all__ = ("extra_grad",)
```