```
from typing import Callable, List
from functools import partial, reduce

import numpy as np
import matplotlib.pyplot as plt
from numpy import ndarray

import time as time

plt.rcParams["figure.figsize"] = (20,10)

%%javascript
IPython.OutputArea.auto_scroll_threshold = 9999;
```

```
<IPython.core.display.Javascript object>
```

```
"""
Находит градиент функции fun в точке x с точностью h.
"""
def grad(fun:Callable[[ndarray], float], x:ndarray, h:float=1e-5) ->
ndarray:
    dim = len(x)
    g = np.zeros(dim)
    step = np.zeros(dim)
    for i in range(dim):
        step[i] = h
        g[i] = (fun(x + step) - fun(x - step)) / (2 * h)
        step[i] = 0
    return g

"""
Находит "частичный" градиент функции, представленный в виде суммы.
Вычисляет сумму производных некоторых аргументов функции fun подряд.

@param fun: список аргументов функции для которых находим частичный
градиент
@param x: точка, в которой вычисляются производные
@param r: массив 2 элементов:
    1 элемент показывает первый аргумент в списке fun, для которого
будет найден градиент.
    2 элемент показывает, для скольких аргументов после первого будет
найден градиент.
@param h: точность вычисленных градиентов

@returns r[1] список, содержащий сумму вычисленных градиентов

"""
def grad_appr(fun:List[Callable[[ndarray], float]], x:ndarray,
r:List=(0, 1), h:float=1e-5) -> ndarray:
    sum = np.zeros(len(x))
```

```
    begin = r[0]
    n = r[1]
    for i in range(n):
        sum += grad(fun[(begin + i) % len(fun)], x, h)
    return sum

"""
Генерирует датасет для множественной линейной регрессии.
Гарантируется, что ожидаемое значение шума равно 0.
Каждый регрессор представляет собой len(b) - 1-мерный вектор.
Как списки регрессоров, так и списки зависимых переменных имеют
размер, равный size.
Для понимания мы можем сказать, что функция генерирует точки,
сдвинутые вверх или вниз относительно некоторой гиперплоскости.

b[1:] - задает в некотором роде нормаль к этой гиперполскости.
b[0] - смещение гиперплоскости вверх или вниз относительно начала
координат
length - длина гиперплоскости.
k - шум, который характеризует, насколько сгенерированные точки будут
отклоняться от гиперплоскости,
которую мы аппроксимируем.
size - количество точек на гиперплоскости.

В итоге получается два массива: точки на гиперплоскости и высоты этих
точек.

@param b:список, характеризующий линейную зависимость между:
регрессорами (независимыми переменными) и зависимой переменной.
b[0] - смещение зависимой переменной

@param k: шум (аргумент ошибки) максимальное абсолютное значение
@param size: количество точек (экземпляров данных) в датасете
@param r: кортеж, описывающий диапазон, в котором будут сгенерированы
регрессоры (функция)

@return список из двух элементов: список регрессоров и список
зависимых значений
"""
def generate_dataset(b:List[float], k:int, size:int=50, r:tuple = (0,
10)) -> List[List[float]]:
    # t += (np.random.rand(len(t)) - 0.5) * 2 * k
    h, b = b[0], b[1:]
    dim = len(b)
    # *[t]*dim is equals to *np.tile(t, dim).reshape((dim, -1))
    regressors = np.random.rand(size, dim) * (r[1] - r[0]) + r[0]
    # Noise
    e = (np.random.rand(len(regressors)) - 0.5) * 2 * k
    dependent_var = np.sum(b * regressors, axis=1) + h + e
    return [regressors, dependent_var]
```

```
"""
Генерирует функцию, минимальная точка которой является решением задачи
линейной регрессии для датасета [t, ft].
Довольно времязатратный процесс.


@param t: список регрессоров
@param ft: список зависимых значений

@возвращает список аргументов сгенерированной функции
(каждый аргумент представляет расстояние от свернутой функции до одной
точки из датасета)
"""
def generate_minimized_fun(t:List[List[float]], ft:List[float]) ->
List[Callable[[List[float]], float]]:
    sum_fun = np.empty(len(t), dtype=partial)
    for i in range(len(t)):
        # Captures t[i], ft[i] and len(t) from current context
        sum_fun[i] = partial(lambda t, ft, l,
                             a: 1 / (2 * l) * np.square(a[0] +
np.sum(a[1:] * t) - ft), t[i], ft[i], len(t))
        # or
        # sum_fun[i] = partial(lambda t, ft, l,
        #a: 1 / (2 * l) * np.square(np.sum(a * np.concatenate(([1],
t))) - ft), t[i], ft[i], len(t))
    np.random.shuffle(sum_fun)
    return sum_fun

"""
Генерирует функцию, минимальная точка которой является решением задачи
линейной регрессии для датасета [t, ft].
Аналогично generate_minimized_fun, но каждый регрессор является 1-
мерным вектором
(т.е. Минимизированная функция имеет два аргумента).
Используется эта функция только тогда, когда необходимо построить
минимизированную функцию.

Функция, сгенерированная generate_minimized_two_variable_fun, отлично
работает, когда аргументы являются массивами,
в отличие от generate_minimized_fun.

@param t: список регрессоров
@param ft: список зависимых значений

@возвращает список аргументов сгенерированной функции
(каждый аргумент представляет расстояние от сгенерированной функции до
одной точки из набора данных)
"""
```

```
def generate_minimized_two_variable_fun(t:List[List[float]],
ft:List[float]) -> List[Callable[[List[float]], float]]:
    sum_fun = np.empty(len(t), dtype=partial)
    for i in range(len(t)):
        # Captures t[i], ft[i] and len(t) from current context
        sum_fun[i] = partial(lambda t, ft, l, a: 1 / (2 * l) *
np.square(a[0] + a[1] * t - ft), t[i], ft[i], len(t))
    np.random.shuffle(sum_fun)
    return sum_fun

"""
Генерирует функцию из аргументов.
Каждый аргумент - это функция.
Каждая функция из аргументов должна принимать равное количество
параметров.

@return функция, представляющая сумму функций из терминов

"""
def fun_from_terms(terms:List[Callable[[List[float]], float]]) ->
Callable[[List[float]], float]:
    return reduce(lambda f1, f2: lambda x: f1(x) + f2(x), terms)

class StandartScaler:
    """
    Предлагает функциональность для выполнения стандартного
масштабирования.
    Идеально работает, когда данные генерируются функцией
generate_dataset.
    @param t: список экземпляров данных для которых вычисляем среднее
значение и отклонение

    """
    def __init__(self, t:ndarray):
        self.mean = np.mean(t, axis=0)
        self.deviation = np.std(t, axis=0)

    """
    Масштабирует объекты (регрессоры) в экземплярах данных.
    Среднее значение и стандартное отклонение после масштабирования
равны 0 и 1 соответственно.

    @param t: список экземпляров данных
    @return список экземпляров данных с масштабируемыми объектами
    """
    def scale(self, t:ndarray) -> ndarray:
        return (t - self.mean) / self.deviation

    """
```

```
    Выполнение обратного масштабирования.

    @param t: список масштабируемых экземпляров данных
    @return список экземпляров данных перед масштабированием

    """
    def scale_reverse(self, t:ndarray) -> ndarray:
        return t * self.deviation + self.mean

    """
    Вычисляет минимальную точку функции с немасштабированными
экземплярами данных,
    используя минимальную точку функции с масштабированными
экземплярами данных.

    @param min_point: масштабированная минимальная точка
    @return начальная минимальная точка
    """
    def reverse_min_point(self, min_point:ndarray) -> ndarray:
        return np.concatenate([min_point[0] - min_point[1:] *
self.mean / self.deviation,
                               min_point[1:] / self.deviation])

"""
!Устарел!
Масштабирует объекты (регрессоры) в экземплярах данных.
Каждое значение объекта находится в диапазоне [r[0], r[1]] после
масштабирования.
Поскольку набор данных, возвращаемый из generate_dataset, обычно
распределяется внутри каждого объекта,
лучше НЕ использовать этот масштаб, если датасет сгенерирован с
помощью generate_dataset.

@param t: список экземпляров данных
@return список экземпляров данных с масштабируемыми объектами
"""
def min_max_scale(t, r=(0, 1)):
    return (t - np.min(t, axis=0)) / (np.max(t, axis=0) - np.min(t,
axis=0)) * (r[1] - r[0]) + r[0]

"""
Строит линии уровня двух переменных функций, представленных терминами.

@param terms: список аргументов построенной функции (должен быть
результатом generate_minimized_two_variable_fun)
@param points: список точек, в которых строятся линии уровня
@param offset: характеризует масштаб графика

"""
```

```
def plot_path_contours(terms:List[Callable[[List[float]], float]],
                       points:List[List[float]], offset:float=None) ->
None:
    min_point = points[-1]
    ax = plt.figure(figsize=(20, 20)).add_subplot()
    ax.plot(points[:, 0], points[:, 1], 'o-', color = "red")
    color_line = np.zeros((10, 3))
    color_line[:, 1:] = 0.7
    color_line[:, 0] = np.linspace(0, 1, 10)
    fun = fun_from_terms(terms)
    if offset is None:
        offset = np.max(min_point) * 1.2
    ttX = np.linspace(min_point[0] - offset, min_point[0] + offset,
200)
    ttY = np.linspace(min_point[1] - offset, min_point[1] + offset,
200)
    X, Y = np.meshgrid(ttX, ttY)
    plt.title('SGD path and level curves', fontsize=22)
    ax.contour(X, Y, fun([X, Y]), levels=np.sort(np.unique([fun(point)
for point in points])), colors = color_line)

"""
Строит 2-мерный график линейной функции с двумя переменными,
представленной fun_coeffs.
fun_coeffs[0] представляет смещение высоты.
"""
def plot_dataset_and_function(t:List[List[float]], ft:List[float],
fun_coeffs:List[float]) -> None:
    ax = plt.figure().add_subplot()
    ax.plot(t, ft, 'o', markersize=1)
    tt = np.linspace(np.min(t), np.max(t), 1000)
    plt.title('Data set and evaluated function', fontsize=22)
    ax.plot(tt, fun_coeffs[1] * tt + fun_coeffs[0])

"""
Показывает, как SGD приближается к минимальной точке вдоль каждой
координаты
"""
def plot_convergence(points:ndarray) -> None:
    ax = plt.figure().add_subplot()
    plt.title('Convergence plot', fontsize=22)
    ax.plot(np.linspace(0, len(points) - 1, len(points)), points[:,
0], color='red')
    ax.plot(np.linspace(0, len(points) - 1, len(points)), points[:,
1], color='green')

"""
Выводит данные, описывающие эффективность SGD.

@param actual_min: реальная минимальная точка вычисляемой функции
```

```
@param point: путь SGD
"""
def print_result(actual_min:ndarray, points:ndarray,
min_point:float=None) -> None:
    if min_point is None:
        min_point = points[-1]
    print(f'Precision: {actual_min - min_point}')
    print(f'Min point: {min_point}')
    print(f'Iterations: {len(points)}')
    print(f'Path: {points}')

"""
Находит минимум функции sum_fun с использованием стохастического
градиентного спуска (sgd).

@param sum_fun: функция в виде списка аргументов для которых ищем
минмимум
@param x: точка начала поиска минимума
@param max_epoch: верхняя граница для количества итераций
(если параметр stop_criteria имеет дефолтное значение, то количество
итераций равно max_epoch)
@param batch_size: количество аргументов sum_fun для поиска градиента
на каждой итерации
@param lr: начальное значение скорости обучения (показывает, насколько
большими будут шаги на каждой итерации)
@param scheduler: как скорость обучения будет меняться во время
итераций
(когда скорость обучения по умолчанию постоянна)
@param stop_criteria: альтернатива критерию остановки max_epoch (когда
значение по умолчанию никогда не выполняется)
@return список точек, представляющих путь SGD. Последняя точка списка
- это найденная минимальная точка.
Количество точек представляет количество эпох до тех пор,
пока не будут выполнены критерии остановки или не будет достигнут
max_epoch

"""
def sgd(sum_fun:List[Callable[[ndarray], float]], x:ndarray,
max_epoch:int, batch_size:int, lr:List[float],
        scheduler:Callable[[List[float]], float]=lambda lr: lr,
        stop_criteria:Callable[[List[float]], bool]=lambda x: False) -
> ndarray:
    lr = np.array(lr)
    points = [x]
    for i in range(1, max_epoch):
        if stop_criteria(x): break
        x = x - 1 / batch_size * scheduler(lr) *
np.array(grad_appr(sum_fun, x, [(i - 1) * batch_size, batch_size]))
        points.append(x)
    return np.array(points)
```

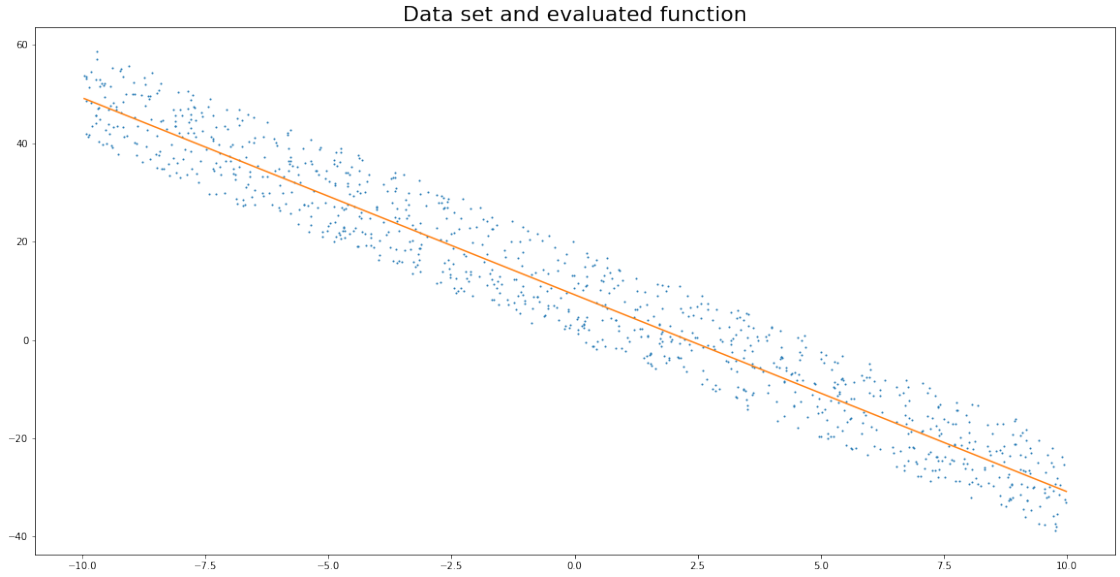
Решение задачи простой линейной регрессии (одномерное пространство регрессоров)

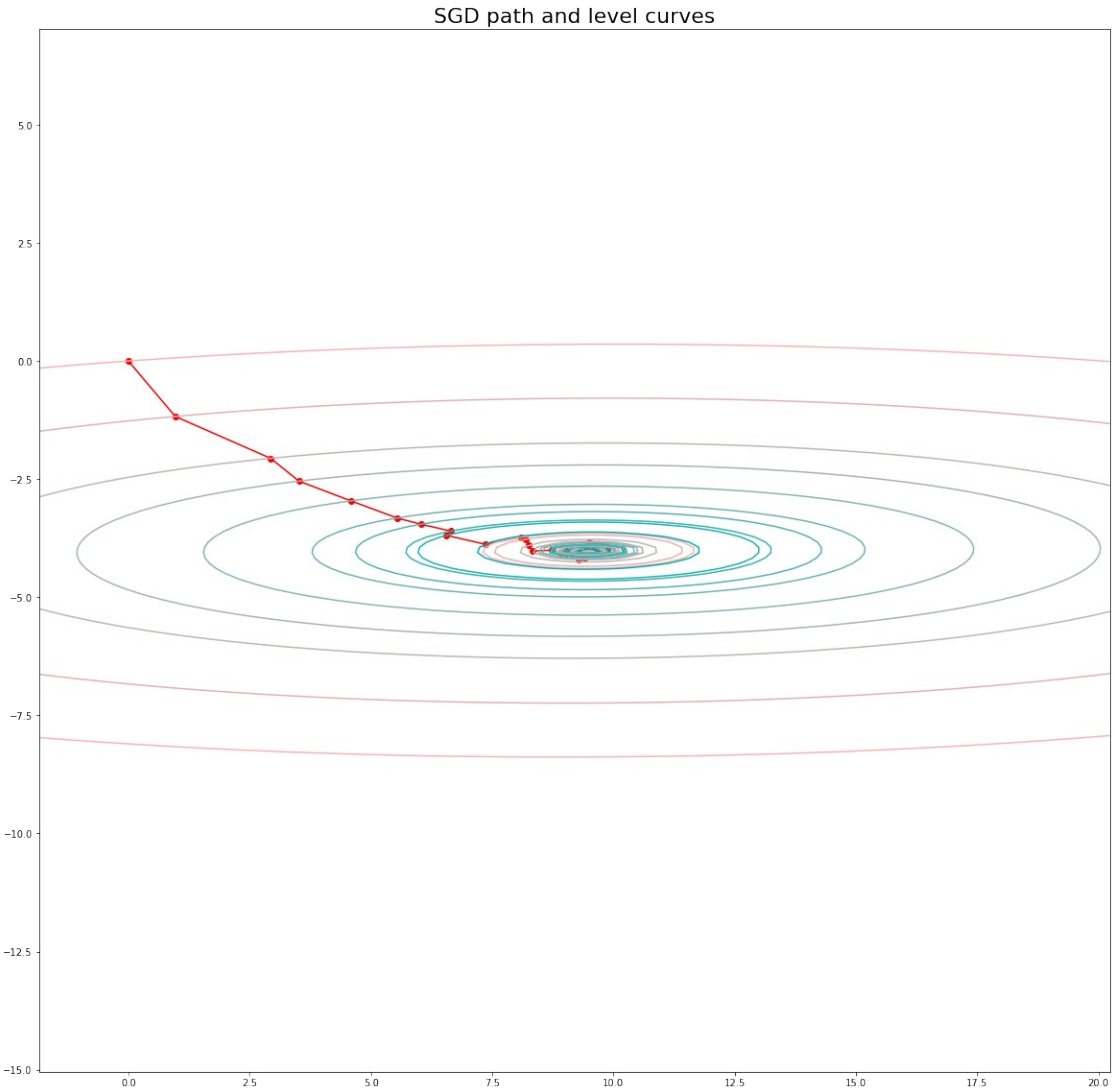
Приближает прямую в двумерном пространстве

```
a = [10, -4]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 50
batch_size = 20
lr = [150, 8]

points = sgd(sum_fun, x, epoch, batch_size, lr, stop_criteria=lambda
x: (np.abs(x - a) < 0.05).all())

print_result(a, points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [0.79852924 0.00266118]
Min point: [ 9.20147076 -4.00266118]
Iterations: 50
Path: [[ 0.          0.        ]
 [ 0.96586862 -1.17330303]
 [ 2.93237569 -2.05980098]
 [ 3.51695238 -2.54245192]
 [ 4.58670857 -2.95980787]
 [ 5.53850299 -3.32280651]
 [ 6.03015222 -3.45395001]
 [ 6.64386949 -3.59957805]
 [ 6.55750818 -3.6900158 ]
 [ 7.35235708 -3.8775436 ]
 [ 8.08728044 -3.73025511]
 [ 8.19607432 -3.76809547]
 [ 8.25610891 -3.89512448]
 [ 8.33247176 -4.02136792]
 [ 8.72054918 -4.00548457]
 [ 8.90243006 -4.05384643]
 [ 9.03470589 -4.01435872]
 [ 9.29449827 -4.03203199]
 [ 9.15121215 -4.08022574]
 [ 9.02670901 -4.0813023 ]
 [ 9.39087672 -4.15767454]
 [ 9.42720805 -4.15576444]
 [ 9.59263853 -3.96305689]
 [ 9.88228759 -4.01164382]
 [ 9.81098211 -4.03492236]
```

```
[ 9.79637074 -4.06388878]
[ 9.86582657 -3.90756132]
[ 9.72347527 -3.95239563]
[ 9.8695404  -3.9896745 ]
[ 9.80343796 -4.03161511]
[ 9.7379151  -4.03749797]
[ 9.46175163 -4.05135844]
[ 9.30858131 -4.08302653]
[ 9.16558041 -4.11965571]
[ 8.9869778  -4.01490304]
[ 9.00810335 -4.01605413]
[ 9.19702687 -4.09522185]
[ 9.2821247  -4.19169781]
[ 9.19431866 -4.11476935]
[ 9.43294832 -4.02192151]
[ 9.60043547 -3.9926637 ]
[ 9.62067973 -3.90577652]
[ 9.43494565 -3.92756067]
[ 9.48550227 -3.87564831]
[ 9.77174734 -3.89012903]
[ 9.49819462 -3.84343978]
[ 9.78368733 -3.96035228]
[ 9.64096171 -3.96871038]
[ 9.64710681 -4.04930022]
[ 9.20147076 -4.00266118]]
```





Решение задачи линейной регрессии (для двумерного пространства регрессоров)

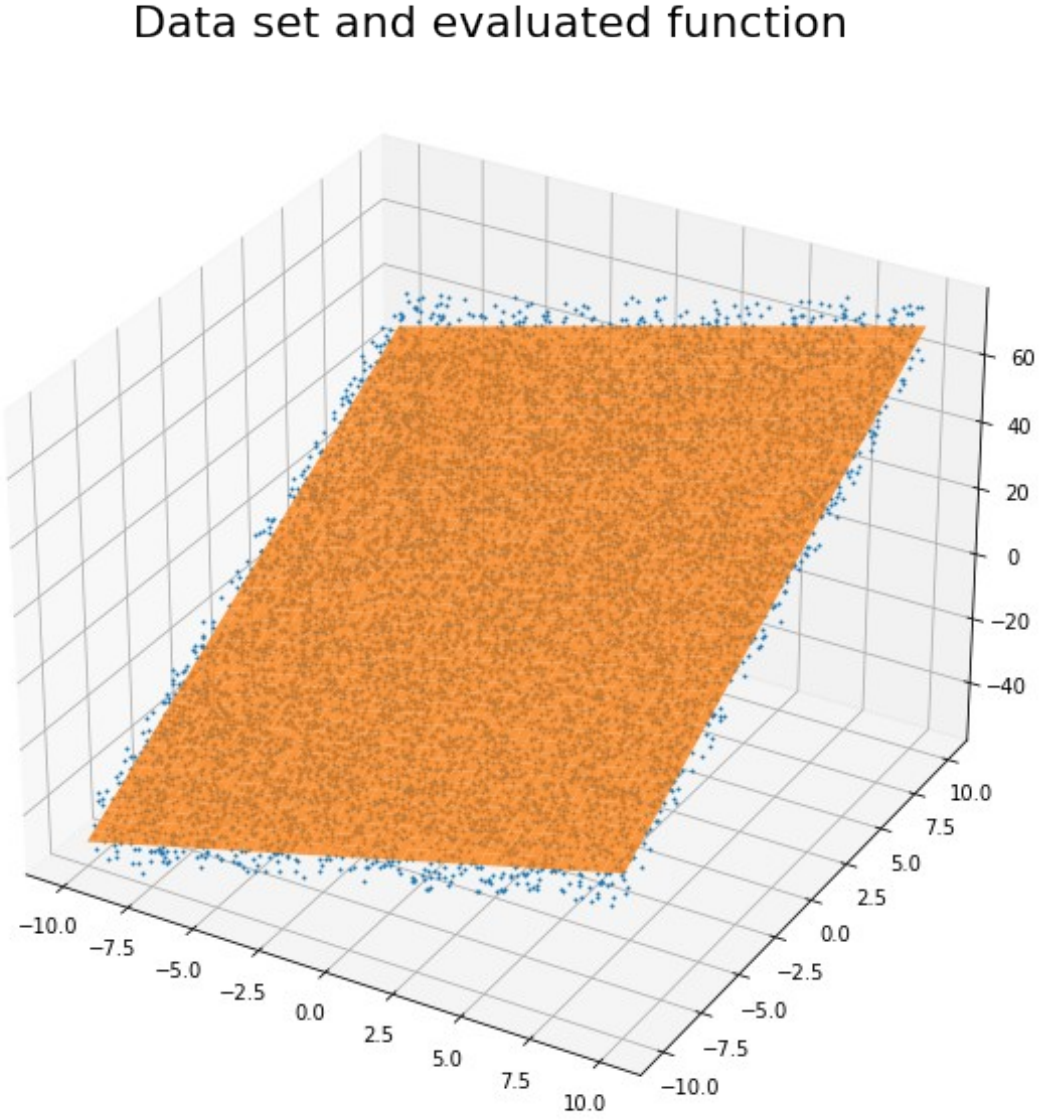
Приближает плоскость в трёхмерном пространстве

```
a = [10, 2, 4]
t, ft = generate_dataset(a, 10, 10000, (-10, 10))
sum_fun = generate_minimized_fun(t, ft)

# SGD params
x = np.zeros(3)
epoch = 50
batch_size = 20
lr = [1500, 40, 80]

points = sgd(sum_fun, x, epoch, batch_size, lr)
min_point = points[-1]
```

```
print_result(a, points)
ax = plt.figure().add_subplot(projection='3d')
# Plots dataset
plt.title('Data set and evaluated function', fontsize=22)
ax.plot(t[:, 0], t[:, 1], ft, 'o', markersize=1)
# Plots approximated function
tt = np.linspace(-10, 10, 1000)
X, Y = np.meshgrid(tt, tt)
ax.plot_surface(X, Y, min_point[0] + min_point[1] * X + min_point[2] *
Y, alpha=0.8)
```

```
Precision: [-0.00104547 -0.08991121  0.01914037]
Min point: [10.00104547  2.08991121  3.98085963]
Iterations: 50
Path: [[ 0.          0.          0.        ]
 [ 2.27266271  0.5243606   2.066692  ]
 [ 3.2914576   0.7321789   2.72695617]
 [ 4.32594013  0.90031218  3.08889325]
 [ 5.24477199  1.09180571  3.28881495]
 [ 5.86084836  1.15583579  3.59597276]
 [ 6.61769196  1.26480235  3.83084685]
 [ 6.91382072  1.33859767  3.70120024]
 [ 7.10380815  1.50557059  3.74378707]
 [ 7.43433964  1.5672215   3.72170577]
 [ 7.85867387  1.66976869  3.75855967]
 [ 8.01785585  1.64084627  3.84666404]
 [ 8.1143111   1.66502739  3.8609965 ]
 [ 8.58480254  1.769178    4.09109601]
 [ 8.91221869  1.73494756  4.08642096]
 [ 9.25952054  1.8463398   4.19779891]
 [ 9.47047228  1.89891798  4.20954996]
 [ 9.42198159  1.90348533  4.16068567]
 [ 9.56015561  1.92848628  4.09891426]
 [ 9.65307432  1.93041052  4.06478167]
 [ 9.61542739  1.9506629   4.01511404]
 [ 9.65219797  1.92586883  3.99678039]
 [ 9.93352543  1.90706922  3.96665451]
 [10.0964716   1.92070326  3.96774957]
 [10.39641457  1.92655963  3.89243072]
 [10.37399154  1.9325122   3.95259971]
 [10.10188202  1.92241493  3.98755701]
 [10.36769676  2.00504926  4.07295366]
 [10.05475717  2.02812888  3.91468648]
 [ 9.74264974  2.03815952  3.9208162 ]
 [ 9.78962544  2.05130781  4.04906025]
 [ 9.85315147  2.0439247   4.11746238]
 [ 9.58926164  2.05590199  4.0678066 ]
 [ 9.77691089  2.12002389  3.96220628]
 [ 9.70525324  2.07710052  3.93816741]
```

```
 [ 9.70530105  2.10371887  3.96509774]
 [ 9.64615813  2.10242775  3.91934104]
 [ 9.8272457   2.15773191  3.95768076]
 [ 9.89593044  2.1698053   3.91971699]
 [ 9.63268622  2.12257952  3.93840008]
 [ 9.43752393  2.09876332  3.8828756 ]
 [ 9.62532569  2.07123131  3.87573513]
 [10.17760723  2.02857945  3.8064302 ]
 [10.3548327   1.96946524  3.89973412]
 [10.30116387  1.98383256  3.97087314]
 [10.3311063   2.02118185  3.92401391]
 [10.37537614  1.99823408  3.98533942]
 [10.38911621  2.01031972  4.07635277]
 [10.38688763  2.02437208  3.97821805]
 [10.00104547  2.08991121  3.98085963]]

<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x284b2ca15b0>
```



4-мерное пространство регрессоров

```
a = [10, 2, 4, 5, -3]
t, ft = generate_dataset(a, 10, 100000, (-10, 10))
sum_fun = generate_minimized_fun(t, ft)

# SGD params
x = np.zeros(5)
epoch = 50
batch_size = 3
lr = [7500, 120, 160, 200, 120]

points = sgd(sum_fun, x, epoch, batch_size, lr)

print_result(a, points)
```

```
Precision: [ 0.21428793  0.2181485   0.39901551  0.1019059  -
0.37849898]
Min point: [ 9.78571207  1.7818515   3.60098449  4.8980941  -
2.62150102]
Iterations: 50
Path: [[ 0.          0.          0.          0.          0.        ]
 [-2.66580522 -0.06474041  0.15318386  0.38963577 -0.11581588]
 [ 0.7727755   0.55775851  0.56649949  1.29342416 -0.43268909]
 [ 1.22558689  0.55646622  0.69902228  1.7086079  -0.40583853]
 [ 1.80812693  0.75885377  0.81090156  2.01727522 -0.48508018]
 [ 1.22328606  0.91121096  1.04699249  2.30575828 -0.61492198]
 [ 2.6462483   0.90560998  1.2314687   2.68133739 -0.60398123]
 [ 5.00943097  1.32086189  1.63666904  3.29892474 -0.94435324]
 [ 6.05017519  1.3307991   1.71755961  3.41246296 -0.94882151]
 [ 6.97835536  1.38709307  1.750987    3.58423408 -0.97465196]
 [ 8.22992738  1.40381402  1.91312679  3.71413918 -1.07741827]
 [ 9.04050193  1.24803188  2.14384853  3.95089476 -1.22108079]
 [ 9.87638239  1.25828472  2.14297064  4.00199348 -1.25999919]
 [ 9.50371318  1.21560179  2.23476051  4.09903306 -1.20280553]
 [ 9.35203595  1.16144578  2.35427843  4.05757089 -1.27415238]
 [ 9.9183105   1.15663912  2.47105122  4.11346268 -1.35680066]
 [ 9.62670184  1.1502519   2.45750604  4.17162515 -1.38719047]
 [10.62783113  1.2269491   2.52212827  4.0772891  -1.50231449]
 [10.44202971  1.29648284  2.50824246  4.10048112 -1.50765116]
 [ 9.93690961  1.29190135  2.51641902  4.08219231 -1.52904597]
 [10.09230717  1.28902519  2.52851038  4.11387678 -1.52325195]
 [10.50243511  1.24177054  2.74454108  4.22229119 -1.63077369]
 [10.09757395  1.35346716  2.8705655   4.27740698 -1.71337607]
 [10.94577737  1.41415703  2.8816927   4.44361195 -1.84770823]
 [10.4487994   1.33224664  3.01519468  4.60301589 -1.90972795]
 [ 9.98186669  1.31171872  3.08900449  4.63921361 -1.89963743]
 [ 9.56792997  1.31611829  3.13815108  4.66113999 -1.94238836]
 [ 9.91778939  1.48954669  3.27872422  4.7143533  -2.01766467]
 [10.08472016  1.55912445  3.368915    4.69623257 -2.06160822]
 [ 9.87508306  1.52892096  3.35827517  4.85268228 -2.1013893 ]
 [ 9.9946156   1.55961386  3.30171884  4.86044152 -2.17060707]
 [10.21605154  1.57997794  3.34207465  4.84012266 -2.17262098]
 [ 9.88853167  1.6047959   3.31426906  4.83093288 -2.21029822]
 [ 9.72141544  1.62327195  3.29560996  4.84723351 -2.20510596]
 [ 9.80106699  1.5662451   3.36613868  4.82601479 -2.23345243]
 [ 9.79858418  1.5660802   3.37739708  4.86287857 -2.27413335]
 [ 9.3576139   1.61302366  3.34236453  4.962671   -2.28606814]
 [ 9.54885559  1.60811603  3.34767893  5.00791565 -2.29125556]
 [ 9.64970801  1.61258944  3.38966349  5.00202792 -2.26513369]
 [ 9.210525    1.603562    3.40011916  5.05832556 -2.31596592]
 [ 9.61378984  1.63822603  3.43302507  5.08027944 -2.36478154]
 [10.10340996  1.6328679   3.41658386  5.13635913 -2.40611702]
 [ 9.781017    1.65099692  3.4373042   5.0617699  -2.41449699]
 [10.28742095  1.65028067  3.50174216  5.00891221 -2.42872643]
 [ 9.91947729  1.65412288  3.50590593  4.98219468 -2.49947899]
```

```
 [ 9.56537947  1.67220252  3.49291712  5.01274574 -2.50410228]
 [ 9.01578826  1.71492253  3.53451314  5.01879001 -2.56466136]
 [ 9.34819354  1.72613073  3.5989392   4.99335579 -2.59945921]
 [ 9.72239103  1.76984198  3.6455317   4.94508785 -2.57827635]
 [ 9.78571207  1.7818515   3.60098449  4.8980941  -2.62150102]]
```

100-мерное пространство регрессоров

```
n = 101
a = np.random.randint(-10, 10, n)
t, ft = generate_dataset(a, 10, 1000000, (-10, 10))
sum_fun = generate_minimized_fun(t, ft)

# SGD params
x = np.zeros(n)
epoch = 50
batch_size = 20
lr = 120

points = sgd(sum_fun, x, epoch, batch_size, lr)

print_result(a, points)
```

```
Precision: [-10.0604957   -5.75134787  -5.88178095  -8.40935702  -
2.69351432
  -4.45298949  -6.47812089   1.69783594  -4.24134244   3.35147376
   3.25636826  -5.56115614  -0.18041074   1.59639781   2.93107257
   3.70807992  -8.35384525  -7.83503137   2.92613742   1.78652899
   1.97069408   2.46759943  -6.40733572   3.93667283   2.28764031
   4.22878557   4.9578739    7.19758248   7.52135884   7.47590341
  -6.00490414  -6.73689191  -1.20658071  -4.70133858  -0.29220909
  -3.1863318   -0.94498261   4.05474202   7.6795092    5.0022397
   6.55719805   1.0472938    7.41162057  -0.68817663  -2.78016871
  -8.25635078  -3.32462199  -7.85496549  -1.56425238  -5.85561032
  -5.78494599  -2.46777465  -2.39699903  -3.38007051   3.5253096
  -1.67049603   4.56269278  -3.33164024  -8.56205858   0.79530831
  -0.24442769   7.12237103   4.96674115  -0.04610151  -2.46384508
   6.08796628  -5.18339039  -8.86308737  -4.3597265   -3.1059128
   6.80995996  -5.77740355   5.44458356   4.899459     1.22831256
   7.65756876  -2.45498397  -0.05324952   5.29487629  -8.05661162
  -1.89141609   1.80189877   4.38976255  -0.72288434   6.80183219
   0.14274747  -0.32003621   6.1547317   -3.23126691  -6.71808702
  -4.04673648   0.11789799  -4.16630654   4.90761126  -6.68987069
  -7.6960821   -7.4202391   -2.65428684   5.52394991   7.06021487
  -5.34485606]
Min point: [ 0.0604957  -1.24865213 -1.11821905 -1.59064298 -
0.30648568 -0.54701051
 -1.52187911  0.30216406 -0.75865756  0.64852624  0.74363174 -
1.43884386
  0.18041074  0.40360219  1.06892743  1.29192008 -1.64615475 -
2.16496863
```

```
  1.07386258  0.21347101  0.02930592  0.53240057 -1.59266428
1.06332717
  0.71235969  0.77121443  1.0421261   0.80241752  1.47864116
1.52409659
 -0.99509586 -1.26310809  0.20658071 -1.29866142  0.29220909 -
0.8136682
 -0.05501739  0.94525798  1.3204908   0.9977603   1.44280195 -
0.0472938
  1.58837943 -0.31182337 -1.21983129 -1.74364922 -0.67537801 -
2.14503451
  0.56425238 -1.14438968 -1.21505401 -0.53222535 -0.60300097 -
0.61992949
  0.4746904  -0.32950397  0.43730722 -0.66835976 -1.43794142
0.20469169
  0.24442769  1.87762897  1.03325885  0.04610151 -0.53615492
0.91203372
 -0.81660961 -1.13691263 -0.6402735   0.1059128   1.19004004 -
1.22259645
  1.55541644  1.100541    0.77168744  1.34243124 -0.54501603 -
0.94675048
  0.70512371 -1.94338838 -0.10858391  0.19810123  0.61023745 -
0.27711566
  1.19816781 -0.14274747  0.32003621  0.8452683  -0.76873309 -
1.28191298
 -0.95326352 -0.11789799 -0.83369346  1.09238874 -1.31012931 -
1.3039179
 -1.5797609  -0.34571316  2.47605009  1.93978513 -1.65514394]
Iterations: 50
Path: [[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 ...
0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [-7.86765594e-04 -8.85669384e-02 -2.06128841e-02 ...  6.94405321e-02
   7.83415063e-02 -6.12472675e-02]
 [ 2.39607084e-03 -1.03854663e-01 -1.46270157e-01 ...  1.70749101e-01
   1.88051002e-01 -1.16233161e-01]
 ...
 [ 5.80579846e-02 -1.25508877e+00 -9.92982155e-01 ...  2.44023445e+00
   1.82357388e+00 -1.60686696e+00]
 [ 5.54092714e-02 -1.23809412e+00 -1.02775817e+00 ...  2.43274200e+00
   1.87515901e+00 -1.59650627e+00]
 [ 6.04957010e-02 -1.24865213e+00 -1.11821905e+00 ...  2.47605009e+00
   1.93978513e+00 -1.65514394e+00]]
```

Сходимость в зависимости от размера batch

Двумерное пространство регрессоров. Можно заметить, что с повышением размера batch графики отклонений становятся всё плавнее, стремясь к некоторой точке, близкой к нулю.

```
a = [25, 2, 15]
t, ft = generate_dataset(a, 10, 100, (-10, 10))
sum_fun = generate_minimized_fun(t, ft)

# SGD params
x = np.zeros(3)
epoch = 50
lr = [9, 0.5, 1]
scheduler = lambda lr: np.array(lr) * np.exp(-0.01)

n = 70
min_points = []
for batch_size in range(1, n + 1):
    min_points.append(sgd(sum_fun, x, epoch, batch_size, lr)[-1])

min_points = np.array(min_points)
deviation = a - min_points
print(deviation)
print(min_points)

tt = np.linspace(1, n, n)
plt.gca().set_xlabel('Batch size', fontsize=14)
plt.gca().set_ylabel('Deviation from minimum point', fontsize=14)
plt.plot(tt, deviation[:, 0], '-o', color='red')
plt.plot(tt, deviation[:, 1], '-o', color='green')
plt.plot(tt, deviation[:, 2], '-o', color='blue')
```

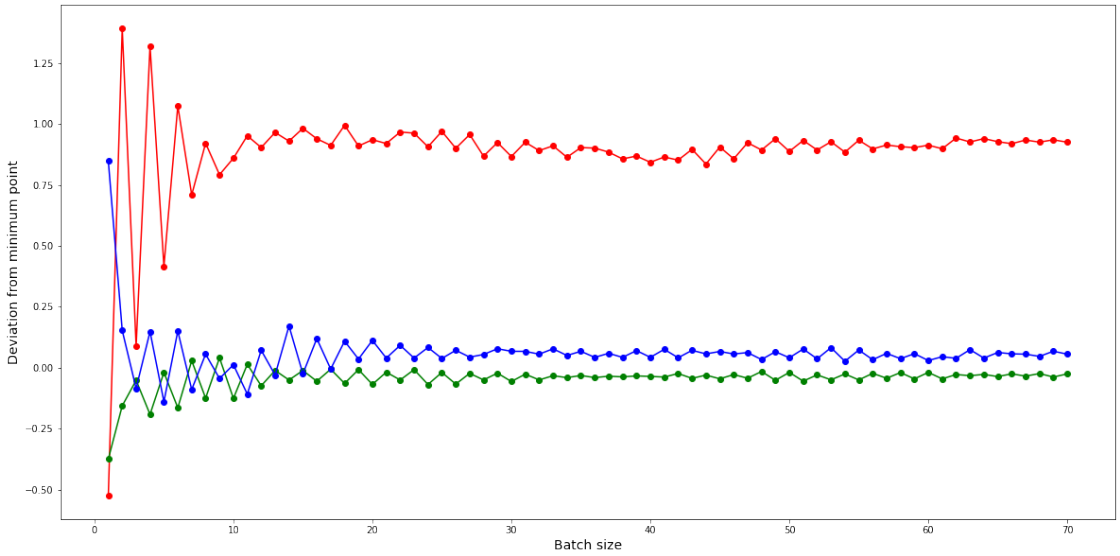
```
[[-0.52558106 -0.37494236  0.84766673]
 [ 1.39306491 -0.15670663  0.15273043]
 [ 0.08889537 -0.05207196 -0.08549064]
 [ 1.31937312 -0.19196055  0.14790788]
 [ 0.41611147 -0.01981539 -0.13967166]
 [ 1.07400566 -0.16462532  0.15006901]
 [ 0.70878217  0.02837918 -0.08999964]
 [ 0.92095391 -0.12509877  0.05526417]
 [ 0.79186105  0.04024962 -0.04459156]
 [ 0.85940988 -0.12635727  0.01173012]
 [ 0.95154306  0.01342605 -0.10866372]
 [ 0.90332082 -0.07438103  0.07128892]
 [ 0.96527763 -0.01151903 -0.03037204]
 [ 0.92957554 -0.0506855   0.17097155]
 [ 0.98214938 -0.01179016 -0.02535629]
 [ 0.93932867 -0.05585212  0.11912262]
 [ 0.91211206 -0.00642754 -0.00567184]
 [ 0.99467901 -0.06479569  0.10875646]
 [ 0.90954818 -0.00805604  0.03507001]
 [ 0.93544784 -0.06880702  0.11184538]
 [ 0.91968747 -0.01860399  0.03918141]
 [ 0.96694508 -0.0507308   0.09160315]
 [ 0.96231006 -0.00806389  0.03911828]
```

```
 [ 0.9061474  -0.06909876  0.08273843]
 [ 0.9697253  -0.01969196  0.03763271]
 [ 0.90021419 -0.06756409  0.07123575]
 [ 0.95661602 -0.02338352  0.04311093]
 [ 0.86778864 -0.05052077  0.05401953]
 [ 0.92368047 -0.02247125  0.07757105]
 [ 0.86648348 -0.05700345  0.0669164 ]
 [ 0.92503858 -0.02671397  0.06679082]
 [ 0.89067931 -0.05038086  0.05526722]
 [ 0.91040088 -0.03310086  0.07728638]
 [ 0.86338006 -0.04101168  0.04984825]
 [ 0.90306844 -0.03212882  0.06706818]
 [ 0.90095193 -0.04085208  0.0421692 ]
 [ 0.88443359 -0.03512763  0.05881364]
 [ 0.85654033 -0.03787779  0.04207724]
 [ 0.86829792 -0.03397694  0.07001184]
 [ 0.84284771 -0.03591915  0.04241009]
 [ 0.86440393 -0.03870073  0.0750152 ]
 [ 0.85160849 -0.0240039   0.0406584 ]
 [ 0.89617107 -0.04411555  0.07098207]
 [ 0.83544323 -0.03051013  0.0563632 ]
 [ 0.90542492 -0.04588785  0.06633597]
 [ 0.8568792  -0.02850438  0.05604622]
 [ 0.92303193 -0.04351626  0.06182853]
 [ 0.89311794 -0.01572185  0.03317414]
 [ 0.93951787 -0.05200977  0.06542167]
 [ 0.8872675  -0.01943311  0.04049885]
 [ 0.93243832 -0.05540117  0.07641555]
 [ 0.8931632  -0.02958034  0.03618232]
 [ 0.92710005 -0.05009689  0.08196953]
 [ 0.8842253  -0.02580906  0.02620934]
 [ 0.93342166 -0.05047231  0.07234225]
 [ 0.89767199 -0.02328082  0.03259967]
 [ 0.91344108 -0.04368126  0.058626  ]
 [ 0.90680502 -0.02008678  0.03666338]
 [ 0.90290422 -0.04620365  0.05745372]
 [ 0.91302014 -0.01937888  0.02905841]
 [ 0.89829097 -0.04602976  0.04485438]
 [ 0.94224532 -0.02824533  0.03811733]
 [ 0.92605469 -0.03292389  0.07395668]
 [ 0.93982581 -0.02777764  0.03914641]
 [ 0.9273017  -0.03671186  0.06225603]
 [ 0.91905968 -0.02541932  0.05660066]
 [ 0.93397129 -0.03491414  0.05525003]
 [ 0.92541567 -0.02423913  0.04610582]
 [ 0.93497238 -0.03864626  0.06735337]
 [ 0.92514256 -0.02557358  0.0552811 ]]
[[25.52558106  2.37494236 14.15233327]
 [23.60693509  2.15670663 14.84726957]
 [24.91110463  2.05207196 15.08549064]
```

```
[23.68062688  2.19196055 14.85209212]
[24.58388853  2.01981539 15.13967166]
[23.92599434  2.16462532 14.84993099]
[24.29121783  1.97162082 15.08999964]
[24.07904609  2.12509877 14.94473583]
[24.20813895  1.95975038 15.04459156]
[24.14059012  2.12635727 14.98826988]
[24.04845694  1.98657395 15.10866372]
[24.09667918  2.07438103 14.92871108]
[24.03472237  2.01151903 15.03037204]
[24.07042446  2.0506855  14.82902845]
[24.01785062  2.01179016 15.02535629]
[24.06067133  2.05585212 14.88087738]
[24.08788794  2.00642754 15.00567184]
[24.00532099  2.06479569 14.89124354]
[24.09045182  2.00805604 14.96492999]
[24.06455216  2.06880702 14.88815462]
[24.08031253  2.01860399 14.96081859]
[24.03305492  2.0507308  14.90839685]
[24.03768994  2.00806389 14.96088172]
[24.0938526   2.06909876 14.91726157]
[24.0302747   2.01969196 14.96236729]
[24.09978581  2.06756409 14.92876425]
[24.04338398  2.02338352 14.95688907]
[24.13221136  2.05052077 14.94598047]
[24.07631953  2.02247125 14.92242895]
[24.13351652  2.05700345 14.9330836 ]
[24.07496142  2.02671397 14.93320918]
[24.10932069  2.05038086 14.94473278]
[24.08959912  2.03310086 14.92271362]
[24.13661994  2.04101168 14.95015175]
[24.09693156  2.03212882 14.93293182]
[24.09904807  2.04085208 14.9578308 ]
[24.11556641  2.03512763 14.94118636]
[24.14345967  2.03787779 14.95792276]
[24.13170208  2.03397694 14.92998816]
[24.15715229  2.03591915 14.95758991]
[24.13559607  2.03870073 14.9249848 ]
[24.14839151  2.0240039  14.9593416 ]
[24.10382893  2.04411555 14.92901793]
[24.16455677  2.03051013 14.9436368 ]
[24.09457508  2.04588785 14.93366403]
[24.1431208   2.02850438 14.94395378]
[24.07696807  2.04351626 14.93817147]
[24.10688206  2.01572185 14.96682586]
[24.06048213  2.05200977 14.93457833]
[24.1127325   2.01943311 14.95950115]
[24.06756168  2.05540117 14.92358445]
[24.1068368   2.02958034 14.96381768]
[24.07289995  2.05009689 14.91803047]
```

```
 [24.1157747   2.02580906 14.97379066]
 [24.06657834  2.05047231 14.92765775]
 [24.10232801  2.02328082 14.96740033]
 [24.08655892  2.04368126 14.941374  ]
 [24.09319498  2.02008678 14.96333662]
 [24.09709578  2.04620365 14.94254628]
 [24.08697986  2.01937888 14.97094159]
 [24.10170903  2.04602976 14.95514562]
 [24.05775468  2.02824533 14.96188267]
 [24.07394531  2.03292389 14.92604332]
 [24.06017419  2.02777764 14.96085359]
 [24.0726983   2.03671186 14.93774397]
 [24.08094032  2.02541932 14.94339934]
 [24.06602871  2.03491414 14.94474997]
 [24.07458433  2.02423913 14.95389418]
 [24.06502762  2.03864626 14.93264663]
 [24.07485744  2.02557358 14.9447189 ]]
```

```
[<matplotlib.lines.Line2D at 0x284af232490>]
```



Решение задачи простой линейной регрессии с использованием standart scaler

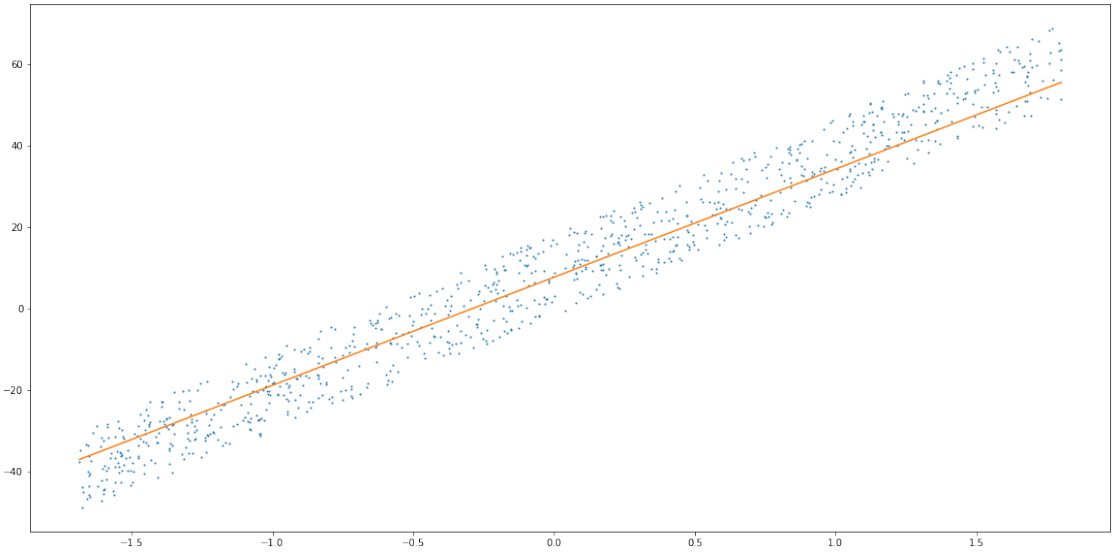
Можно заметить, что после scaling функция изменяется более равномерно по всем измерениям. Значительное улучшение сходимости. Отсутствие необходимости подбирать lr по каждой переменной.

```
a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
scaler = StandartScaler(t)
t = scaler.scale(t)
sum_fun = generate_minimized_two_variable_fun(t, ft)
```

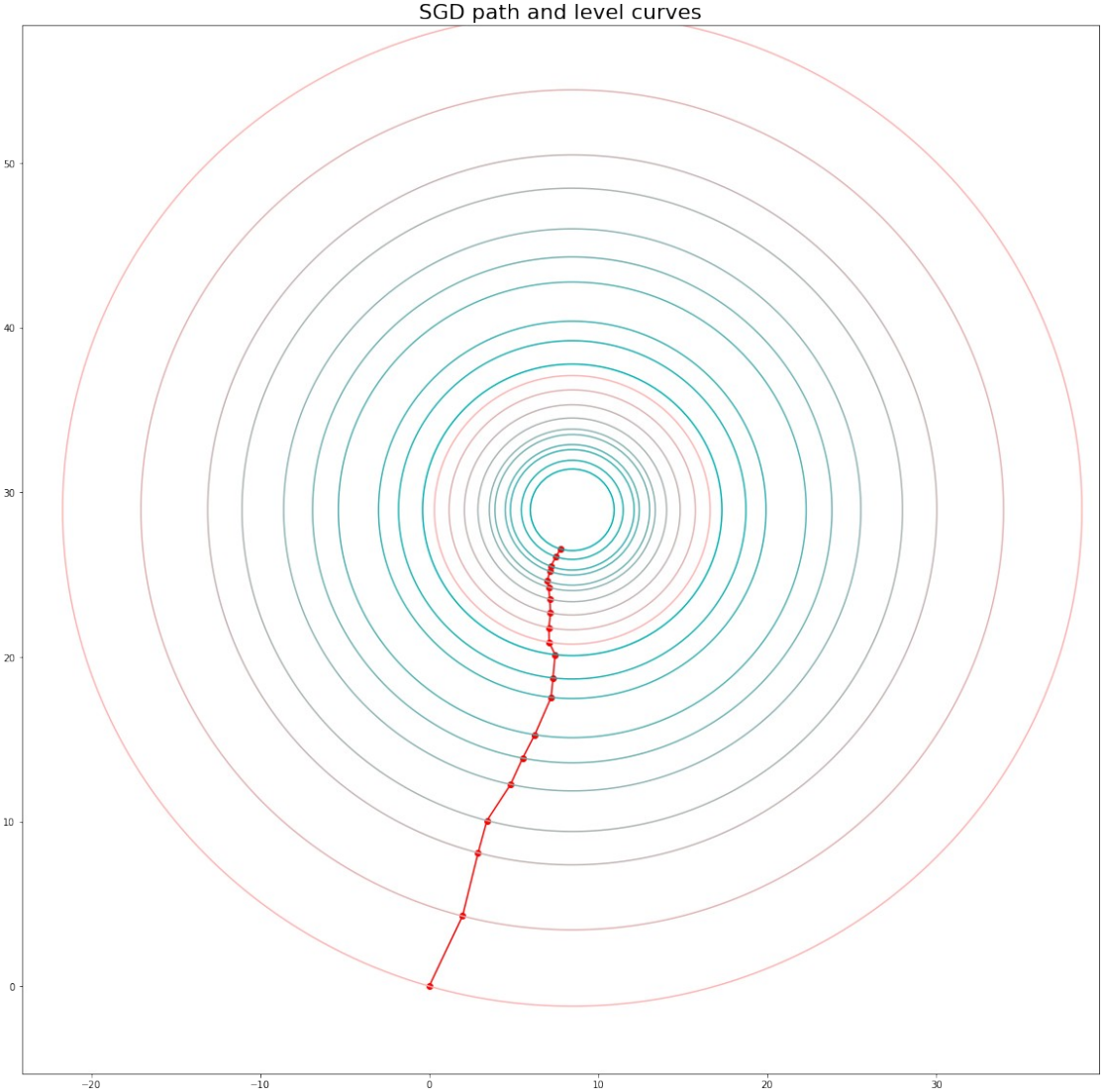
```python
# SGD params
x = np.zeros(2)
epoch = 20
batch_size = 20
lr = 120

points = sgd(sum_fun, x, epoch, batch_size, lr)

print_result(a, points, scaler.reverse_min_point(points[-1]))
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [0.73683582 0.36265478]
Min point: [9.26316418 4.63734522]
Iterations: 20
Path: [[ 0.          0.        ]
 [ 1.9739464   4.26211935]
 [ 2.89764076  8.10167754]
 [ 3.42003814 10.06078513]
 [ 4.81868067 12.25896091]
 [ 5.53342331 13.85437834]
 [ 6.24916706 15.2771561 ]
 [ 7.21555451 17.54491474]
 [ 7.32888597 18.72467249]
 [ 7.45207476 20.14074136]
 [ 7.10210062 20.89355792]
 [ 7.08595092 21.78264109]
 [ 7.17865059 22.68259064]
 [ 7.13708138 23.51667767]
 [ 7.06982444 24.23078775]
 [ 6.98307849 24.60536942]
 [ 7.13169369 25.19927956]
 [ 7.19410262 25.50283854]
 [ 7.49973594 26.08170556]
 [ 7.77038081 26.55311491]]
```

Data set and evaluated function



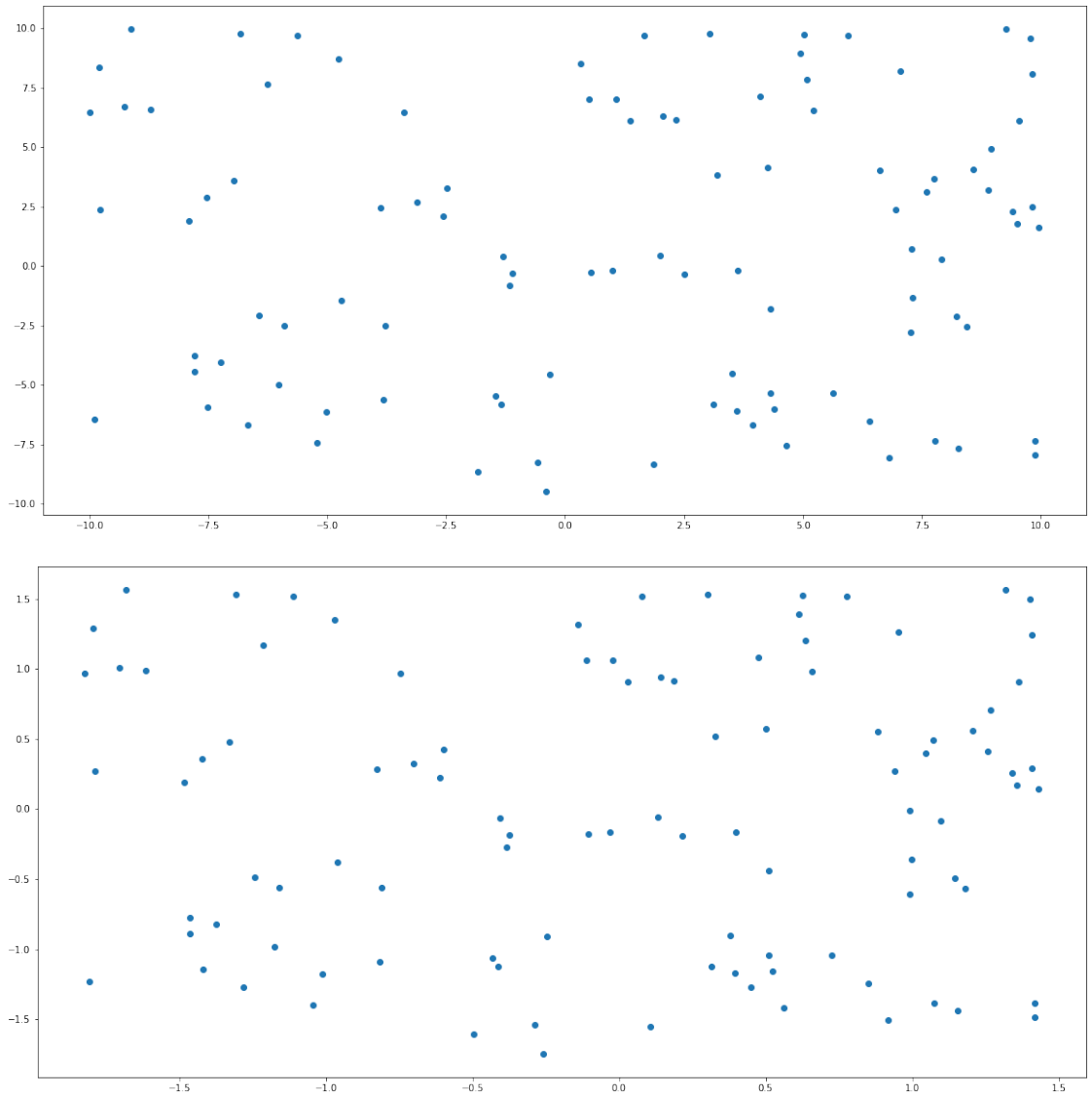
SGD path and level curves



Сравнение до и после standart scaling

```
a = [10, -3, 5]
t = generate_dataset(a, 3, 100, (-10, 10))[0]

ax = plt.figure().add_subplot()
ax.plot(t[:, 0], t[:, 1], 'o')

scaler = StandartScaler(t)
t = scaler.scale(t)

ax = plt.figure().add_subplot()
ax.plot(t[:, 0], t[:, 1], 'o')
```

```
[<matplotlib.lines.Line2D at 0x284add9b7c0>]
```

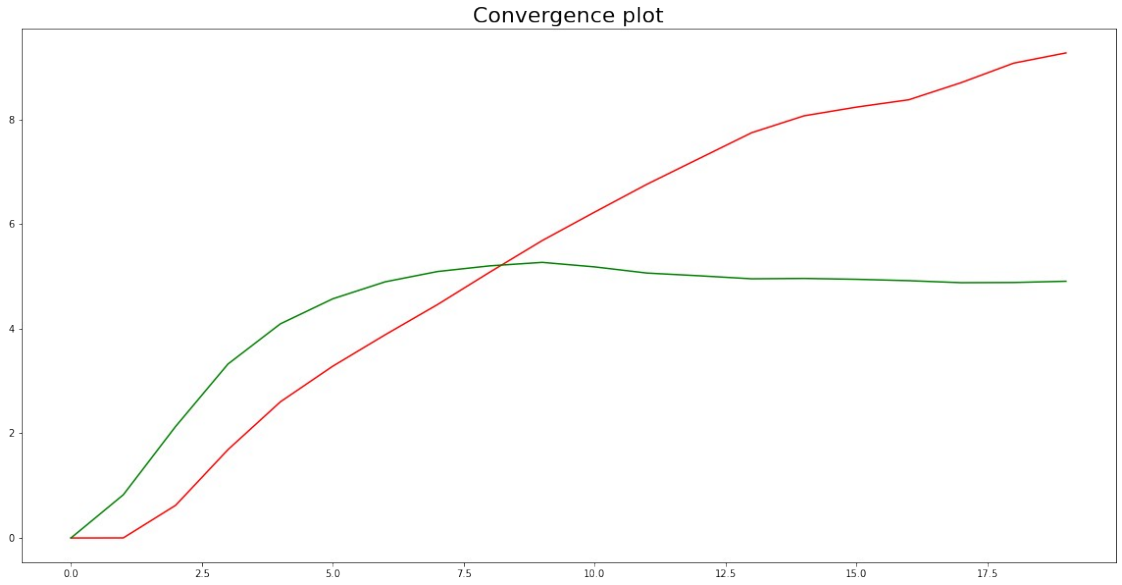


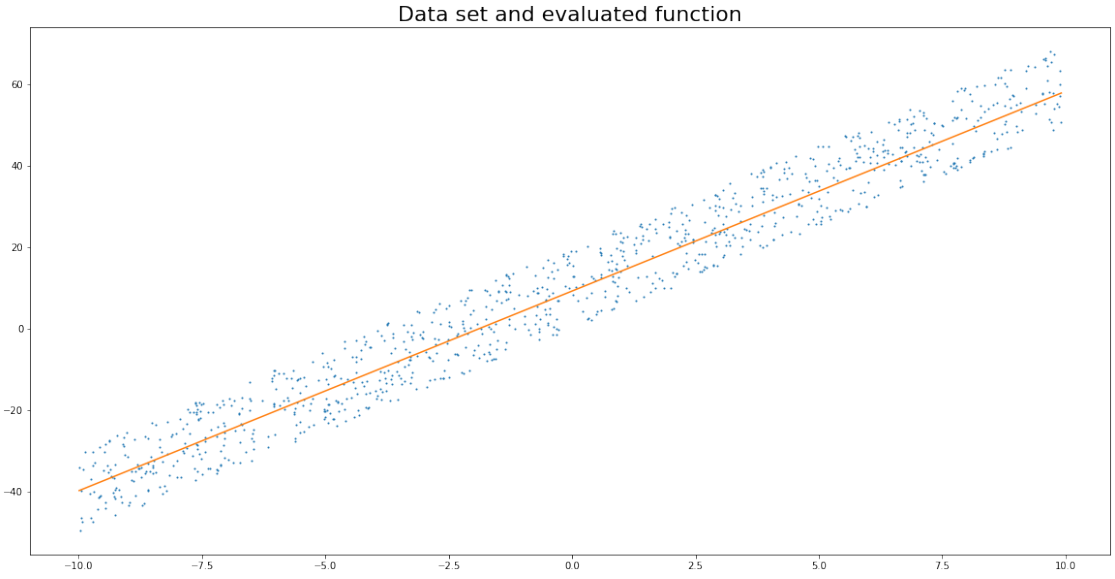
SGD with momentum

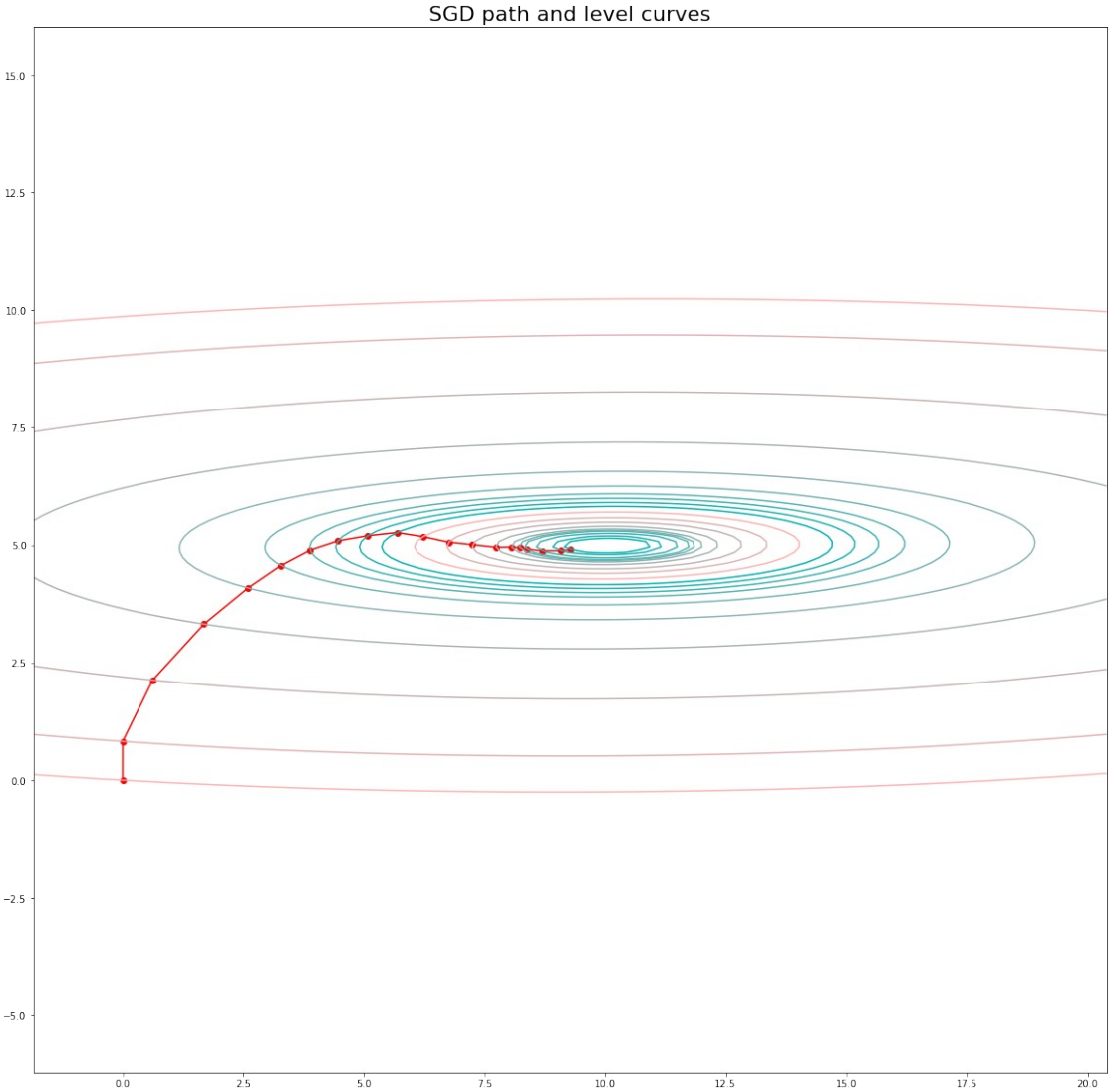
```
"""
Находит минимум функции, используя стохастический градиентный спуск с
импульсом.

Отличие от обычного SGD в том, что добавился параметр b.
Теперь при вычислении градиента в новой точке можно учитывать градиент
во всех предыдущих.
Чем дальше номер точки от текущего номера, тем меньший вклад градиент
в ней вносит в текущий градиент.
Такой подход призван предотвратить излишние колебания минимизируемой
функции вблизи её минимума.

@param b: вес градиента в предыдущих точках
"""
def sgd_momentum(sum_fun:List[Callable[[ndarray], float]],
                 x:ndarray, max_epoch:int, batch_size:int,
                 lr:List[float], b:List[float] = (0),
                 scheduler:Callable[[List[float]], float] = lambda lr:
lr,
                 stop_criteria:Callable[[List[float]], bool]=lambda x:
False) -> ndarray:
    lr = np.array(lr)
    points = [x]
    g = np.zeros(len(x))
    for i in range(1, max_epoch):
        if stop_criteria(x): break
        g = b * g + scheduler(lr) * np.array(grad_appr(sum_fun, x, [(i
- 1) * batch_size, batch_size]))
        x = x - 1 / batch_size * g
        points.append(x)
    return np.array(points)

a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 20
batch_size = 20
lr = [60, 5]
b = [0.5]

points = sgd_momentum(sum_fun, x, epoch, batch_size, lr, b)

print_result(a, points)
plot_convergence(points)
```

```
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [0.72889582 0.09317853]
Min point: [9.27110418 4.90682147]
Iterations: 20
Path: [[0.00000000e+00 0.00000000e+00]
 [1.64115491e-03 8.25707021e-01]
 [6.25166091e-01 2.13302569e+00]
 [1.68972998e+00 3.32601021e+00]
 [2.60267585e+00 4.09332864e+00]
 [3.28369908e+00 4.57149816e+00]
 [3.88347135e+00 4.89588272e+00]
 [4.46158508e+00 5.09262267e+00]
 [5.08180820e+00 5.20146466e+00]
 [5.68487753e+00 5.26787161e+00]
 [6.22851627e+00 5.18130579e+00]
 [6.76120633e+00 5.06334466e+00]
 [7.25414302e+00 5.01117309e+00]
 [7.74615782e+00 4.95369643e+00]
 [8.06873712e+00 4.96039124e+00]
 [8.23505935e+00 4.94419245e+00]
 [8.37587734e+00 4.91772740e+00]
 [8.70303417e+00 4.87946077e+00]
 [9.07473561e+00 4.88304324e+00]
 [9.27110418e+00 4.90682147e+00]]
```



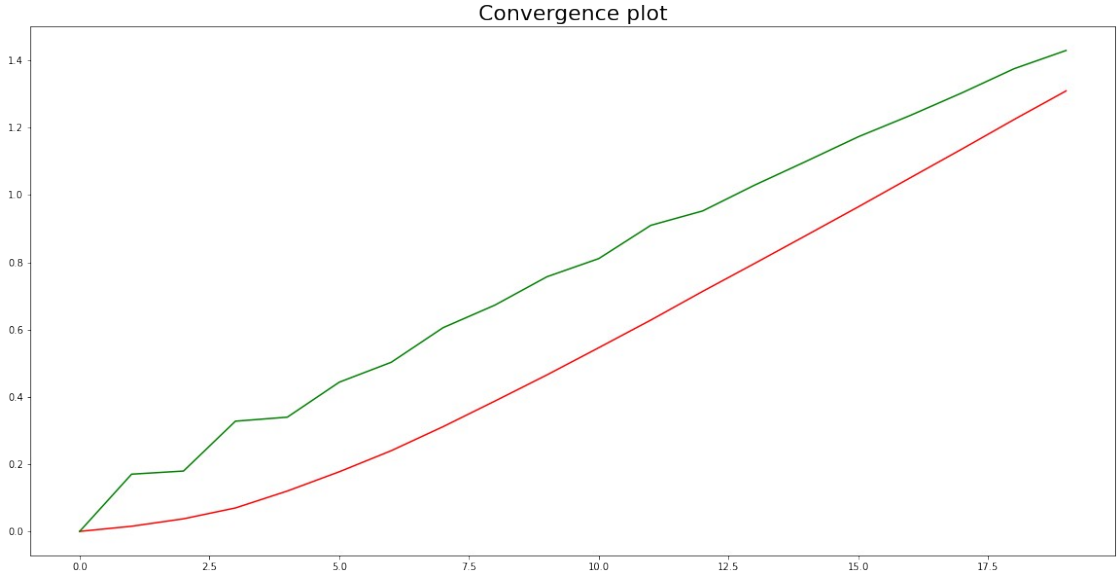

Data set and evaluated function
60
40
20
0
−20
−40
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

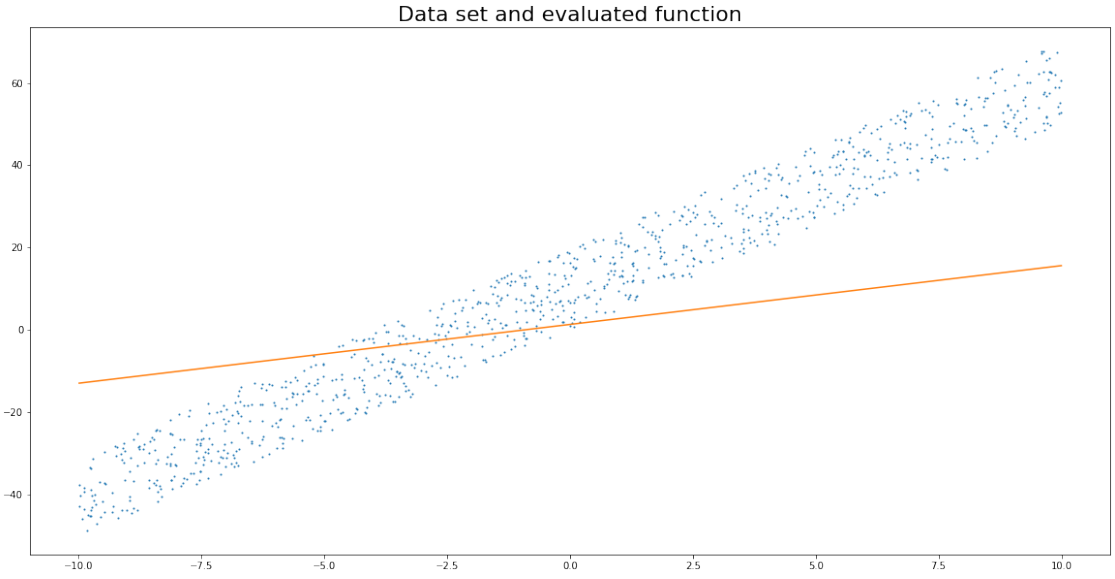

SGD path and level curves
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

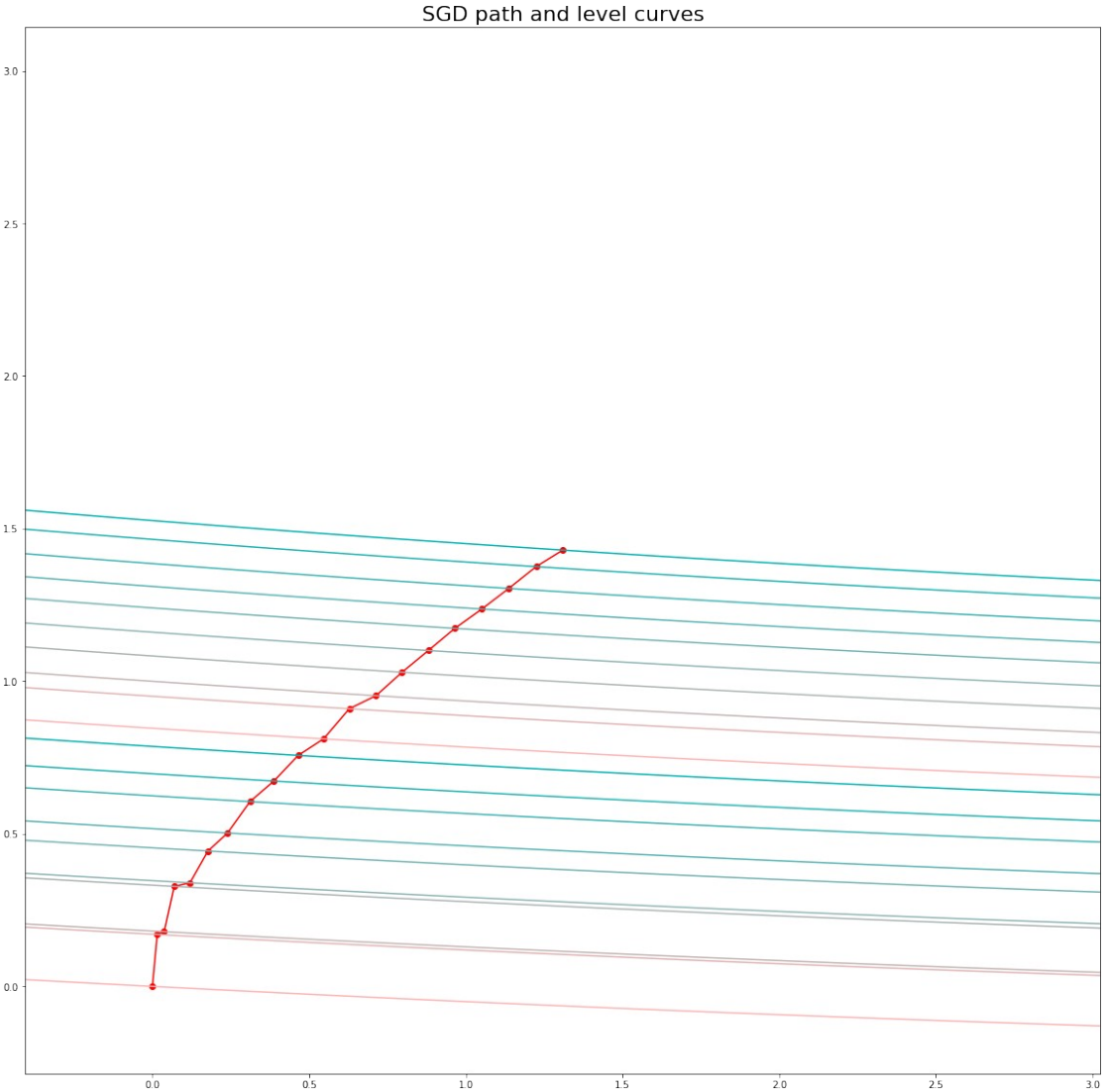
Nesterov SGD

```
"""
Находит минимум функции, используя стохастический градиентный спуск с
использованием алгоритма Нестерова.

Аналогична SGD с импульсом, но в добавок вычисляет градиент на каждом
шаге, не в точке x, а в точке x - b * g,
где g - направление спуска на прошлом шаге.

По неизвестным причинам такой спуск работает намного хуже остальных.
Он либо сходится крайне медленно, либо чрезвычайно быстро расходится.
При этом грань между сходимость и расходимостью очень тонкая.

@param b: вес градиента в предыдущих точках
"""
def sgd_nesterov(sum_fun:List[Callable[[ndarray], float]], x:ndarray,
max_epoch:int, batch_size:int,
                 lr:List[float], b:List[float] = (0),
scheduler:Callable[[List[float]], float] = lambda lr: lr,
                 stop_criteria:Callable[[List[float]], bool]=lambda x:
False) -> ndarray:
    lr = np.array(lr)
    points = [x]
    g = np.zeros(len(x))
    for i in range(1, max_epoch):
        if stop_criteria(x): break
        g = b * g + scheduler(lr) * np.array(grad_appr(sum_fun, x - b
* g, [(i - 1) * batch_size, batch_size]))
        x = x - 1 / batch_size * g
        points.append(x)
    return np.array(points)

a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 20
batch_size = 60
lr = [2, 1]
b = [0.9]

points = sgd_nesterov(sum_fun, x, epoch, batch_size, lr, b)

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [8.69068046 3.5704453 ]
Min point: [1.30931954 1.4295547 ]
Iterations: 20
Path: [[0.         0.        ]
 [0.01531249 0.17008356]
 [0.03743359 0.1794051 ]
 [0.0696716  0.32761118]
 [0.12000897 0.33956744]
 [0.17720267 0.44353241]
 [0.23980239 0.5026053 ]
 [0.31131803 0.60552397]
 [0.38750599 0.6727617 ]
 [0.46512556 0.75688863]
 [0.54611278 0.81096593]
 [0.62804712 0.90966588]
 [0.71342375 0.95256527]
 [0.79642135 1.0292378 ]
 [0.88039741 1.1005051 ]
 [0.96492877 1.17313898]
 [1.05117256 1.23645424]
 [1.13702658 1.30390724]
 [1.22437385 1.37540463]
 [1.30931954 1.4295547 ]]
```



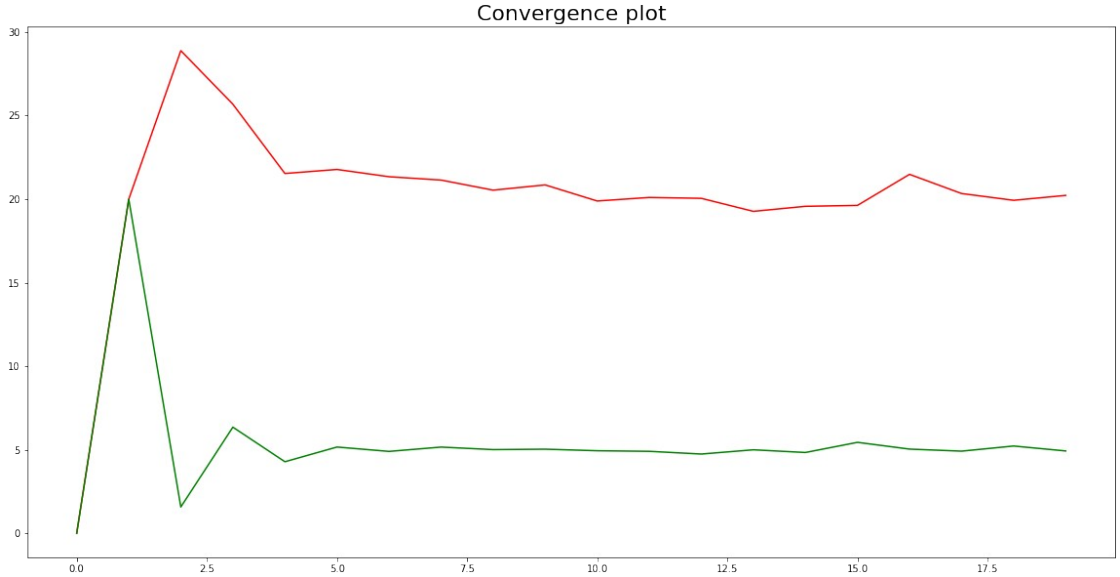

Data set and evaluated function
60
40
20
0
−20
−40
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


SGD path and level curves
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0

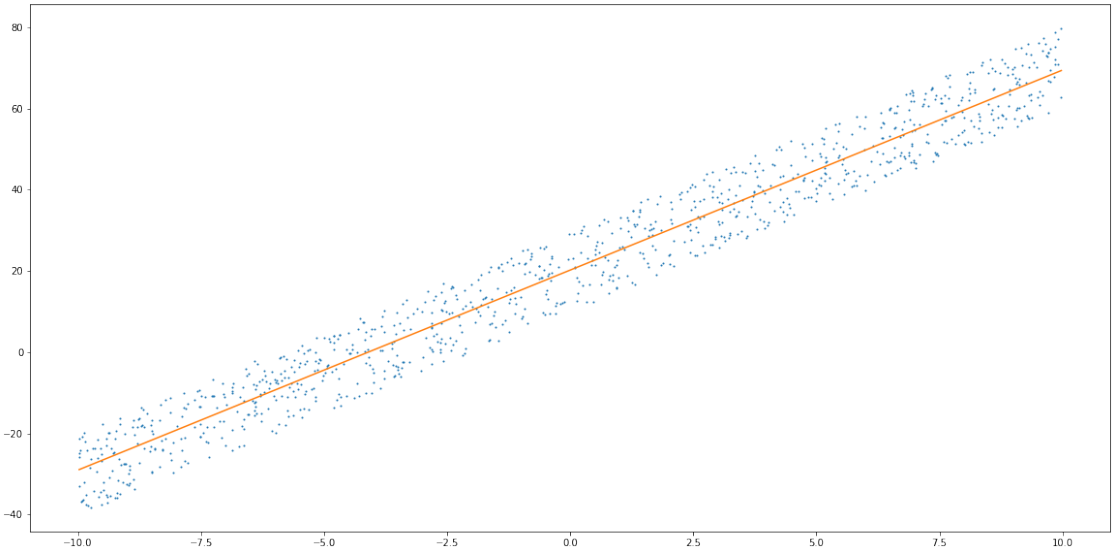
SGD with AdaGrad

```
"""
Находит минимум функции, используя стохастический градиентный спуск с
AdaGrad (адаптивный градиент).

Теперь величина шага накапливается в переменной v. С помощью
накопленной величины редактируется размер шага в направлении
антиградиента на текущей итерации. Само направление при этом
выбирается стандартным образом.

Уменьшает колебания. Даёт возможность использовать одинаковый lr по
всем направлениям даже без использования scaler.
"""
def sgd_adagrad(sum_fun:List[Callable[[ndarray], float]], x:ndarray,
max_epoch:int, batch_size:int, lr:List[float],
scheduler:Callable[[List[float]], float] = lambda lr: lr,
stop_criteria:Callable[[List[float]], bool]=lambda x: False) ->
ndarray:
    lr = np.array(lr)
    points = [x]
    v = np.zeros(len(x))
    for i in range(1, max_epoch):
        if stop_criteria(x): break
        g = np.array(grad_appr(sum_fun, x, [(i - 1) * batch_size,
batch_size]))
        v += np.square(g)
        x = x - 1 / (np.sqrt(v) + 1e-8) * 1 / batch_size *
scheduler(lr) * g
        points.append(x)
    return np.array(points)

a = [20, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 20
batch_size = 50
lr = 1000

points = sgd_adagrad(sum_fun, x, epoch, batch_size, lr)

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

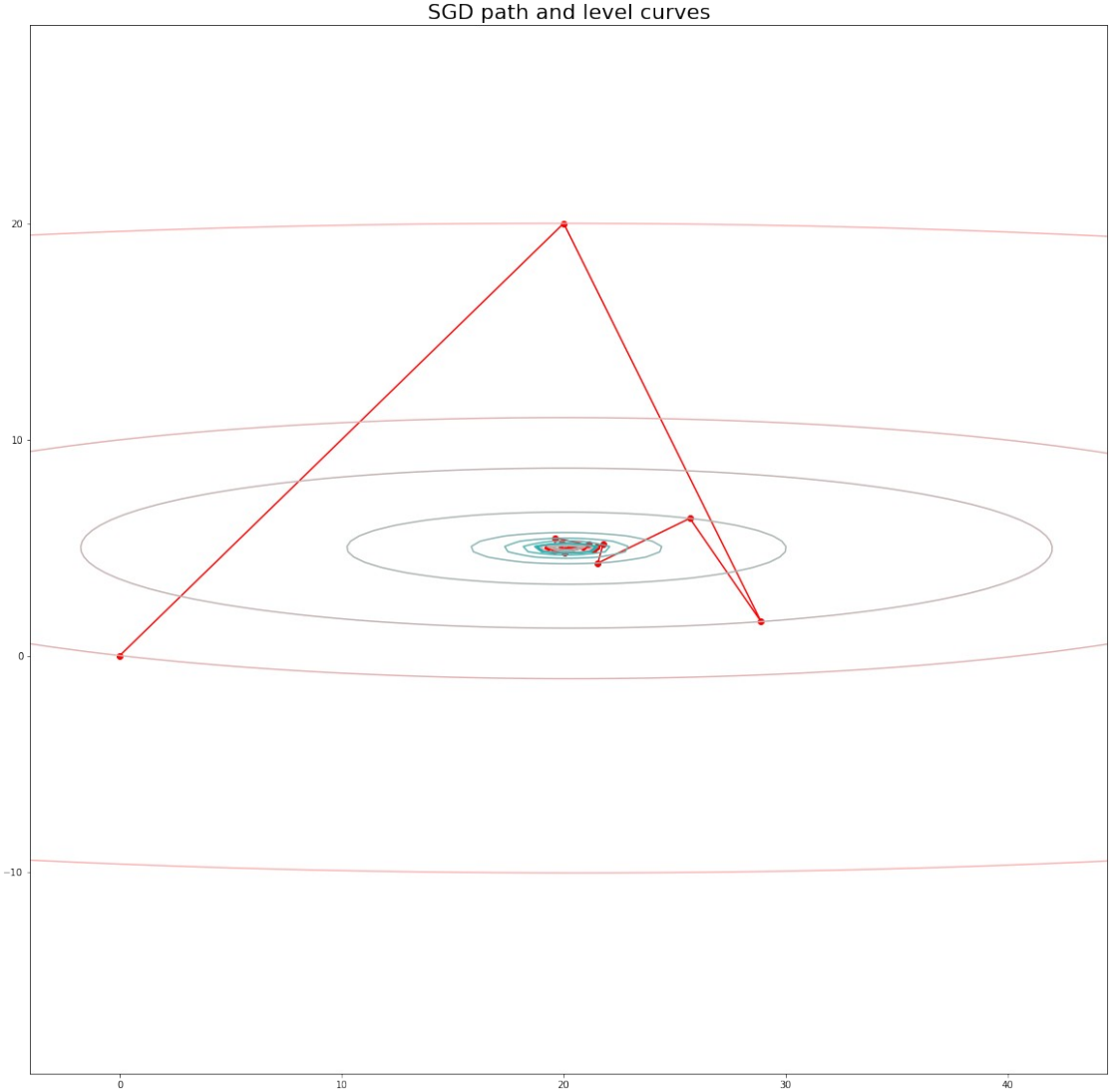
```
Precision: [-0.2257624   0.07349888]
Min point: [20.2257624   4.92650112]
Iterations: 20
Path: [[ 0.          0.        ]
 [19.99999982 19.99999998]
 [28.8919064   1.57158521]
 [25.68478579  6.35129514]
 [21.53649583  4.2805339 ]
 [21.77870214  5.16085784]
 [21.34347845  4.89916517]
 [21.14292538  5.15790025]
 [20.53923137  5.00691947]
 [20.85290809  5.03270676]
 [19.88973876  4.93912859]
 [20.10358284  4.90416   ]
 [20.04962863  4.74142943]
 [19.27077418  4.99557248]
 [19.57124148  4.83413306]
 [19.62854998  5.44154198]
 [21.48570198  5.03730515]
 [20.33758624  4.91456804]
 [19.93199298  5.22157767]
 [20.2257624   4.92650112]]
```



Data set and evaluated function



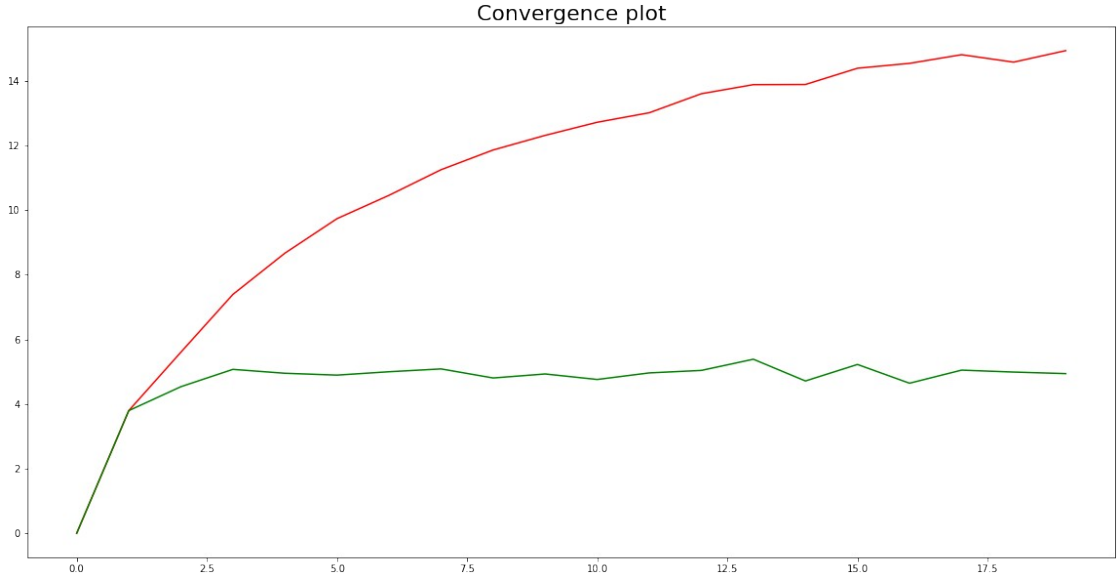
SGD path and level curves

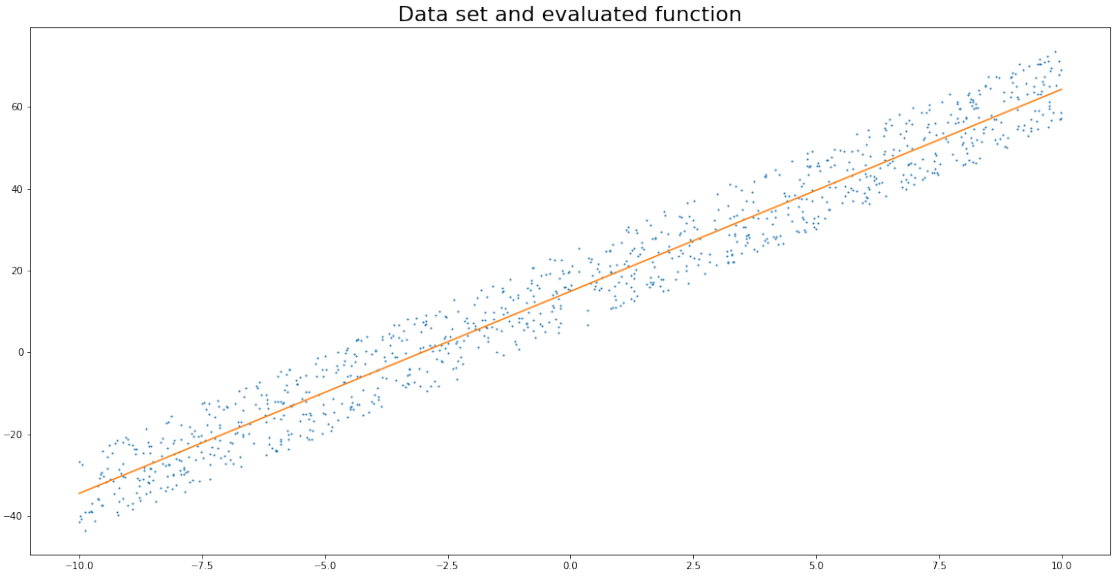


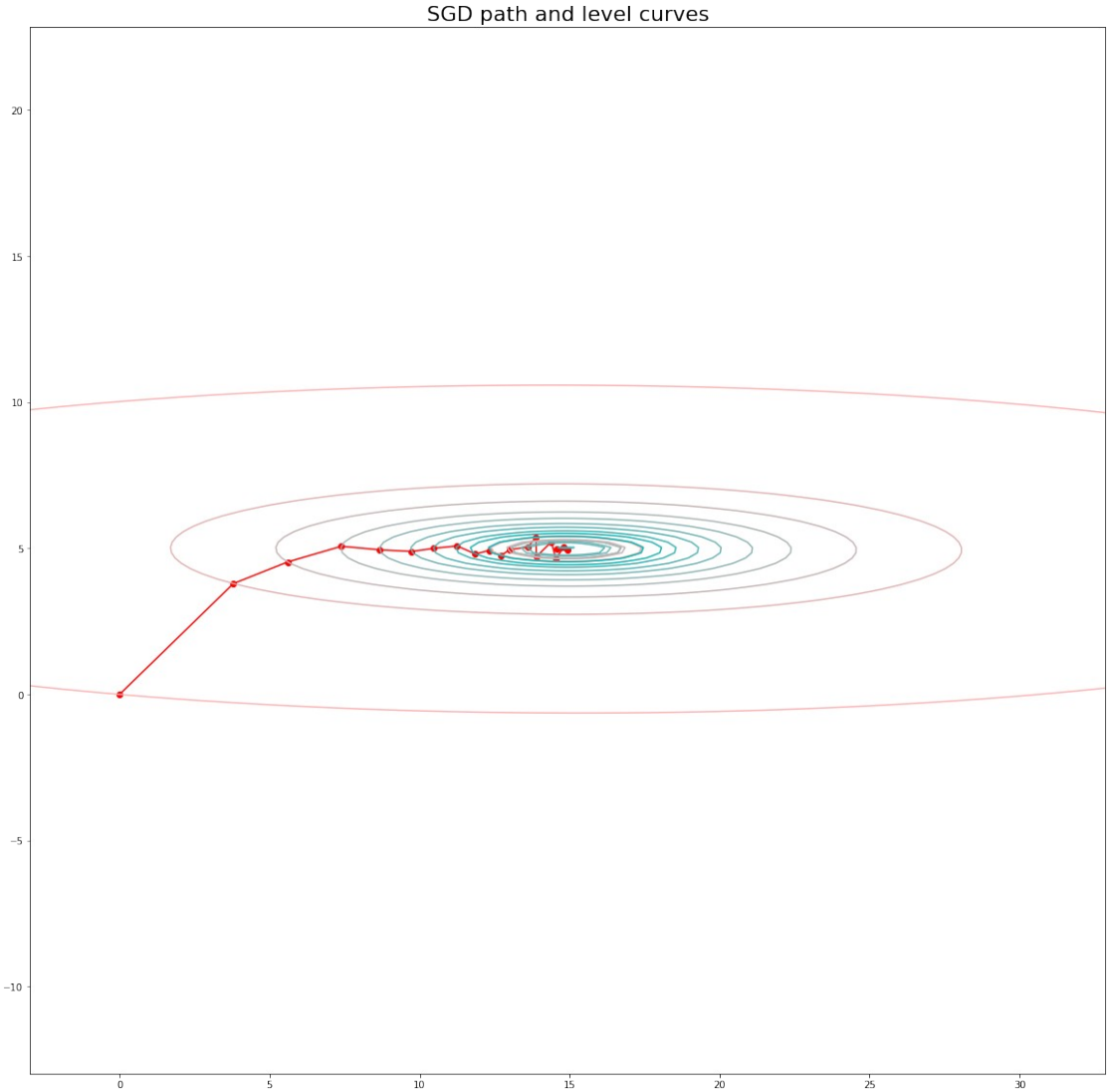
SGD with RMSProp

```
"""
Находит минимум функции, используя стохастический градиентный спуск с
помощью rmsprop.

Алгоритм действует аналогично AdaGrad, но теперь шаг накапливается с
инерцией, изменяясь, как скользящее среднее шагов на прошлых
итерациях.

@param b: вес шагов (евклидова норма градиента) в предыдущих точках

"""
def sgd_rmsprop(sum_fun:List[Callable[[ndarray], float]], x:ndarray,
max_epoch:int, batch_size:int, lr:List[float], b:List[float],
scheduler:Callable[[List[float]], float] = lambda lr: lr,
stop_criteria:Callable[[List[float]], bool]=lambda x: False) ->
ndarray:
    lr = np.array(lr)
    b = np.array(b)
    points = [x]
    v = 0
    for i in range(1, max_epoch):
        if stop_criteria(x): break
        g = np.array(grad_appr(sum_fun, x, [(i - 1) * batch_size,
batch_size]))
        v = b * v + (1 - b) * np.square(g)
        x = x - 1 / (np.sqrt(v) + 1e-8) * 1 / batch_size *
scheduler(lr) * g
        points.append(x)
    return np.array(points)


a = [15, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 20
batch_size = 50
lr = 60
b = 0.9

points = sgd_rmsprop(sum_fun, x, epoch, batch_size, lr, b)

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [0.06311931 0.05934928]
Min point: [14.93688069  4.94065072]
Iterations: 20
Path: [[ 0.          0.        ]
 [ 3.79473307  3.79473318]
 [ 5.60120587  4.53501478]
 [ 7.39092172  5.07182902]
 [ 8.66889791  4.95136235]
 [ 9.7358699   4.89366382]
 [10.45910831  4.99850262]
 [11.25149221  5.0857861 ]
 [11.86197141  4.8054138 ]
 [12.31458647  4.92922768]
 [12.72030281  4.75918297]
 [13.01453785  4.9623685 ]
 [13.60305848  5.03948282]
 [13.88439414  5.38547463]
 [13.88965659  4.71264372]
 [14.39214449  5.22372318]
 [14.54324327  4.64320423]
 [14.81146432  5.0482203 ]
 [14.58076978  4.98856651]
 [14.93688069  4.94065072]]
```




Data set and evaluated function
60
40
20
0
−20
−40
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

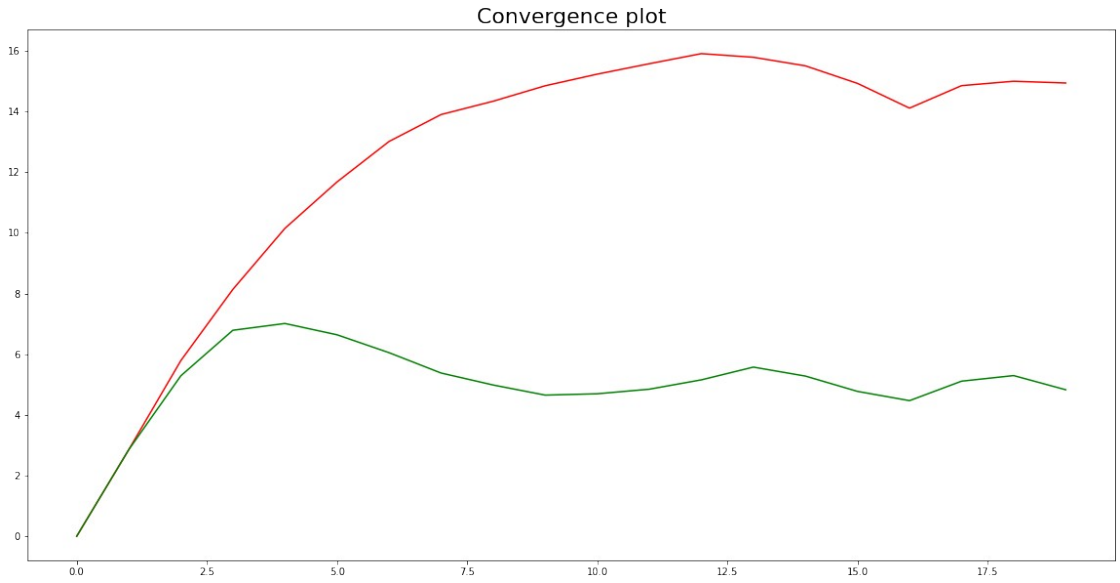

SGD path and level curves
20
15
10
5
0
−5
−10
0
5
10
15
20
25
30

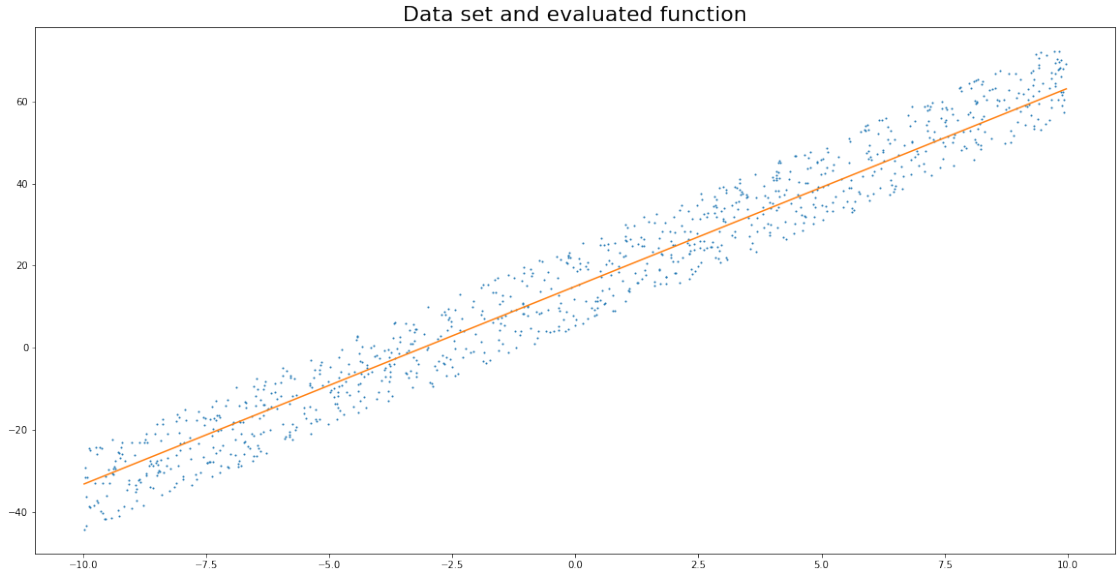
SGD with Adam

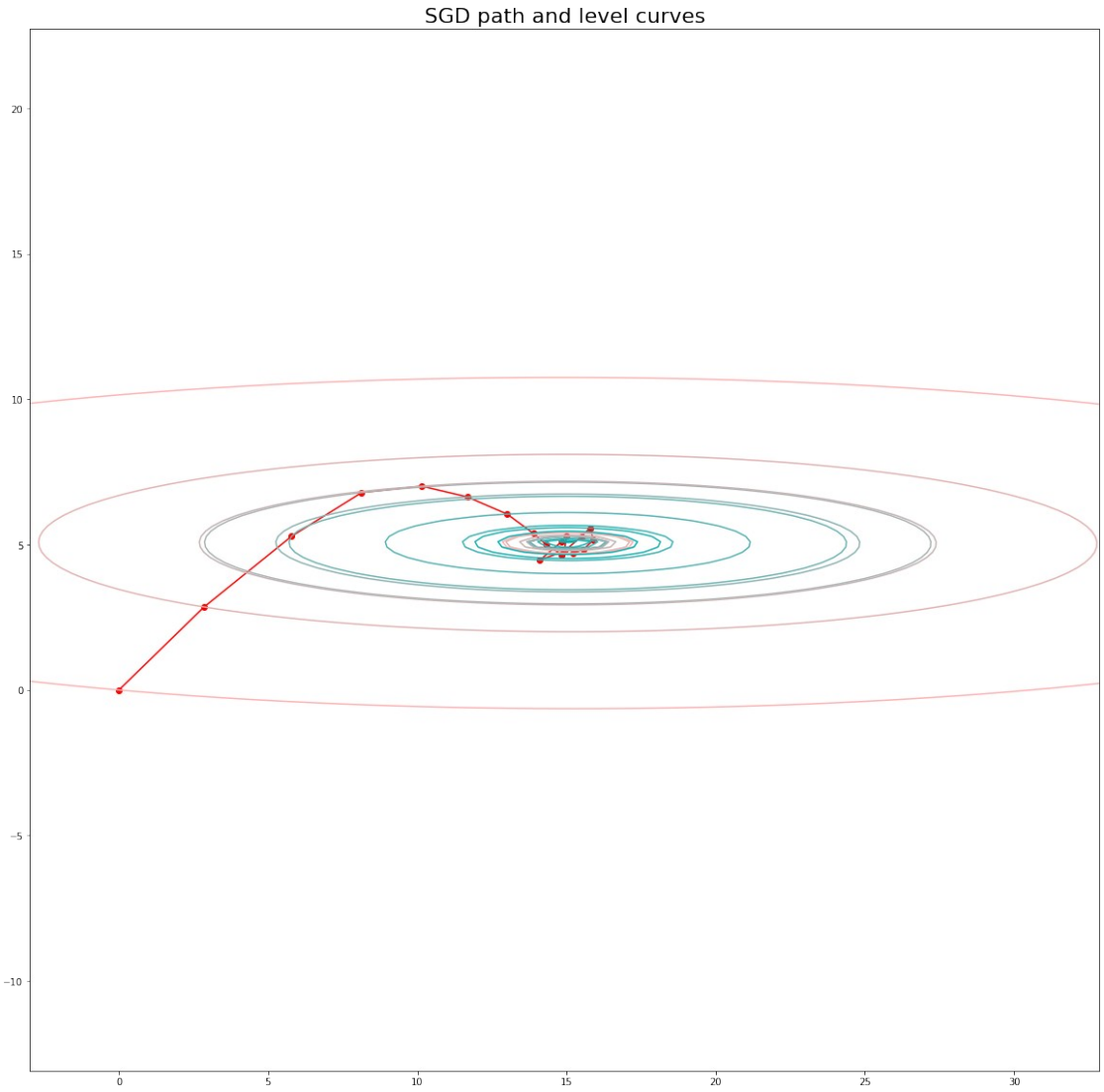
```
"""
Находит минимум функции, используя стохастический градиентный спуск с
помощью adam.

Adam вобрал в себя идеи RMSProp и SGD with momentum. Теперь и величина
шага, и направление шага корректируются,
как скользящее среднее величины шага и его направления на прошлых
итерациях.
Подобный подход позволяет значительно уменьшить колебания спуска
вблизи минимума.
Более того, этот алгоритм, как и RMSProp,
будет хорошо сходиться даже в условиях отсутствия предварительной
нормировки данных,
с одинаковым lr по всем направлениям (features).
Возможно, этот алгоритм на даёт лучшей сходимости всегда,
но он определённо надеждней многих прочих в большинстве случаев.

@param b1:вес градиента в предыдущих точках
@param b2: вес шагов (евклидова норма градиента) в предыдущих точках
"""
def sgd_adam(sum_fun:List[Callable[[ndarray], float]], x:ndarray,
max_epoch:int, batch_size:int, lr:List[float], b1:List[float],
b2:List[float], scheduler:Callable[[List[float]], float] = lambda lr:
lr, stop_criteria:Callable[[List[float]], bool]=lambda x: False) ->
ndarray:
    lr = np.array(lr)
    b1, b2 = np.array(b1), np.array(b2)
    points = [x]
    m = 0
    v = 0
    for i in range(1, max_epoch):
        if stop_criteria(x): break
        g = np.array(grad_appr(sum_fun, x, [(i - 1) * batch_size,
batch_size]))

        m = b1 * m + (1 - b1) * g
        v = b2 * v + (1 - b2) * np.square(g)

        m = m / (1 - np.power(b1, i))
        v = v / (1 - np.power(b2, i))

        x = x - 1 / (np.sqrt(v) + 1e-8) * 1 / batch_size *
scheduler(lr) * m
        points.append(x)
    return np.array(points)

a = [15, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
```

```python
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 20
batch_size = 20
lr = 60
b1 = 0.5
b2 = 0.7
scheduler = lambda lr: lr * np.exp(-0.05)

points = sgd_adam(sum_fun, x, epoch, batch_size, lr, b1, b2,
scheduler=scheduler)

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [0.06319344 0.17198972]
Min point: [14.93680656  4.82801028]
Iterations: 20
Path: [[ 0.          0.        ]
 [ 2.85368813  2.85368826]
 [ 5.78703663  5.28680254]
 [ 8.13546271  6.78831913]
 [10.1464411   7.01131046]
 [11.67501594  6.63983292]
 [13.00904512  6.04975581]
 [13.89720418  5.37921779]
 [14.33592678  4.98420372]
 [14.8452676   4.650863  ]
 [15.22852993  4.69494988]
 [15.57197491  4.84576279]
 [15.90441201  5.15334355]
 [15.78492563  5.5740062 ]
 [15.50239341  5.27789988]
 [14.9241973   4.77372837]
 [14.1086353   4.46844038]
 [14.8498815   5.11053543]
 [14.9953003   5.29439874]
 [14.93680656  4.82801028]]
```


Convergence plot
16
14
12
10
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Data set and evaluated function
60
40
20
0
−20
−40
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



Траектория спуска обычного SGD для одномерного пространства регрессоров

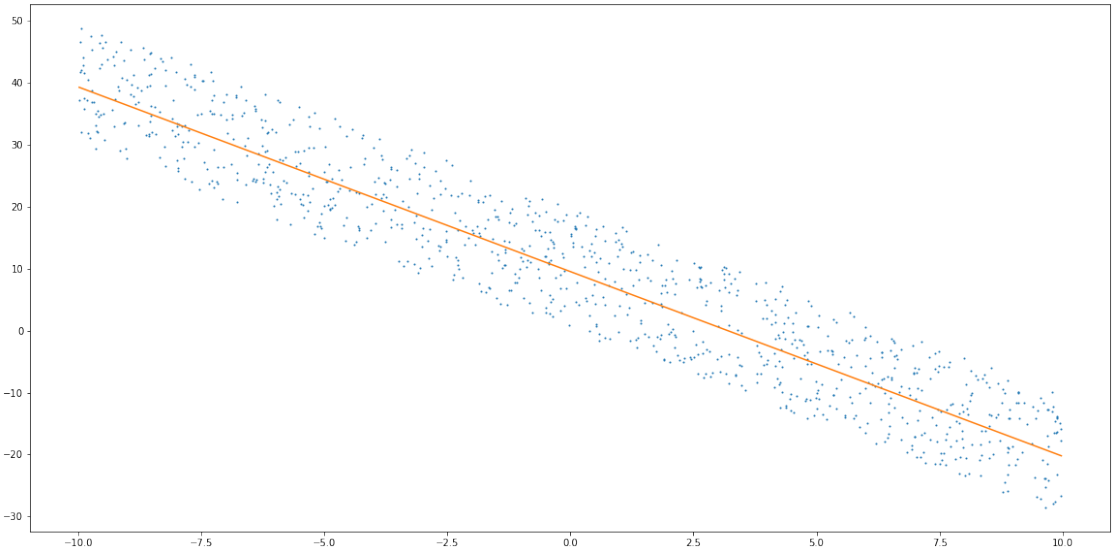
На графике также отображены линии равного уровня минимизируемой функции

```
a = [10, -3]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
epoch = 40
batch_size = 20
lr = [70, 3]

points = sgd(sum_fun, x, epoch, batch_size, lr)
```

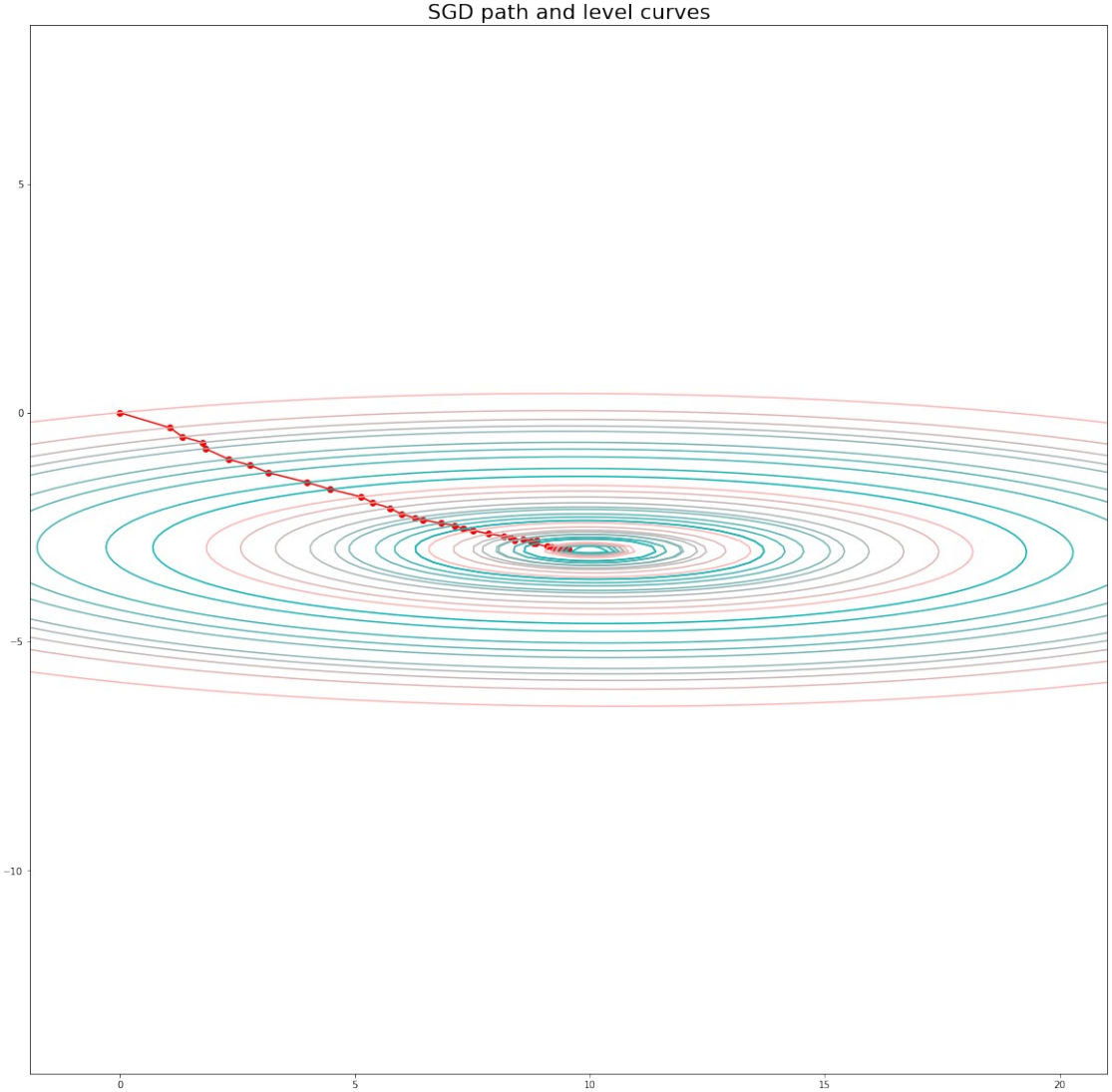
```
print_result(a, points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Precision: [ 0.44865146 -0.01766662]
Min point: [ 9.55134854 -2.98233338]
Iterations: 40
Path: [[ 0.          0.        ]
 [ 1.06625767 -0.32980566]
 [ 1.33376688 -0.52838664]
 [ 1.76095249 -0.65296757]
 [ 1.82319685 -0.78630215]
 [ 2.31863501 -1.01961792]
 [ 2.76521525 -1.14751583]
 [ 3.15318469 -1.30920377]
 [ 3.98573344 -1.52320252]
 [ 4.48088064 -1.6760709 ]
 [ 5.13798647 -1.84067161]
 [ 5.37770508 -1.96828715]
 [ 5.74798278 -2.08369712]
 [ 5.9886394  -2.21775305]
 [ 6.2784011  -2.29677029]
 [ 6.44245927 -2.34285009]
 [ 6.82697605 -2.41595664]
 [ 7.12920487 -2.46399798]
 [ 7.3059889  -2.54139421]
 [ 7.30549183 -2.54251548]
 [ 7.5204495  -2.57573281]
 [ 7.84850725 -2.64989188]
 [ 8.16435647 -2.69807449]
 [ 8.32225243 -2.7188128 ]
 [ 8.39659028 -2.78187098]
 [ 8.57248057 -2.75909675]
 [ 8.74973623 -2.80927152]
 [ 8.86420458 -2.78740303]
 [ 8.85126501 -2.84813306]
 [ 8.81701738 -2.85716831]
 [ 9.09699109 -2.92519715]
 [ 9.27559739 -2.95562341]
 [ 9.2540395  -2.95408126]
 [ 9.24936926 -2.95646301]
 [ 9.19709284 -2.94252953]
 [ 9.33154233 -2.97425035]
 [ 9.44419609 -2.96646867]
 [ 9.35408366 -2.97169108]
 [ 9.42228468 -2.97707287]
 [ 9.55134854 -2.98233338]]
```

Data set and evaluated function



SGD path and level curves



Нагрузочное тестирование SGD_Nesterov

```
a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
infinity_epoch = 1000
batch_size = 60
lr = [2, 1]
b = [0.9]
stop_criteria:Callable[[List[float]], bool]=lambda x: np.sum(np.abs(x
- a)) < 0.05

start_time = time.time()
points = sgd_nesterov(sum_fun, x, infinity_epoch, batch_size, lr, b,
stop_criteria=stop_criteria)
print(f'Process time: {(time.time() - start_time) / len(points)}')

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Process time: 0.0042529740232102415
Precision: [0.02879652 0.01841185]
Min point: [9.97120348 4.98158815]
Iterations: 376
Path: [[0.         0.        ]
 [0.03528234 0.20425652]
 [0.07057439 0.18693056]
 [0.11382328 0.36589133]
 [0.17731071 0.3592504 ]
 [0.24891368 0.51035715]
 [0.32471532 0.52277197]
 [0.40347436 0.68178803]
 [0.48873761 0.70460179]
 [0.5774997  0.82700394]
 [0.67203172 0.8695449 ]
 [0.76257287 0.95667663]
 [0.85375328 1.01861266]
 [0.94594097 1.09788471]
 [1.03718773 1.15541438]
 [1.12997163 1.23597353]
 [1.21934163 1.28835596]
 [1.30901325 1.37681247]
 [1.39240199 1.41144636]
 [1.47964965 1.48765131]
 [1.57025905 1.53032025]
 [1.66037762 1.61172666]
```

```
[1.74571348 1.6475736 ]
[1.83037719 1.72867386]
[1.91399329 1.77237043]
[1.9961988  1.83684248]
[2.07829529 1.88104104]
[2.16180012 1.93664851]
[2.24242336 1.9853116 ]
[2.32199979 2.04125942]
[2.40226557 2.09152146]
[2.48162939 2.14152034]
[2.55934065 2.18980714]
[2.63594936 2.24919783]
[2.70769428 2.2943193 ]
[2.77750931 2.33455078]
[2.85003291 2.38950268]
[2.92475356 2.423     ]
[2.9999543  2.47694303]
[3.07327657 2.50714205]
[3.14503935 2.57258907]
[3.21611844 2.59964165]
[3.28761159 2.66005454]
[3.36102673 2.67871385]
[3.43066016 2.73424269]
[3.49858505 2.76266597]
[3.56747516 2.82011408]
[3.63559096 2.83559113]
[3.70390923 2.89924874]
[3.76968113 2.91371419]
[3.835774   2.97993774]
[3.89411674 2.98474832]
[3.95274087 3.03129967]
[4.01590218 3.05077762]
[4.07860488 3.10202801]
[4.13908104 3.11345406]
[4.19824599 3.15444862]
[4.25834782 3.18921597]
[4.31668896 3.21826861]
[4.37441608 3.24986563]
[4.43449588 3.27904902]
[4.49185935 3.30594526]
[4.54820118 3.33642749]
[4.60561441 3.36491902]
[4.66200861 3.39275411]
[4.72034387 3.42508014]
[4.77555501 3.44866569]
[4.82875915 3.48726336]
[4.87717041 3.49841808]
[4.92830688 3.52746614]
[4.98230849 3.54628741]
[5.0366112  3.58181365]
```

```
[5.08732534 3.59127495]
[5.13843185 3.63009658]
[5.18844624 3.64842703]
[5.23763363 3.67612727]
[5.28714277 3.69230196]
[5.33864846 3.71549054]
[5.3876131  3.73515232]
[5.4358535  3.76024996]
[5.48510287 3.78119762]
[5.53381947 3.80177337]
[5.58118769 3.82150533]
[5.62752381 3.85043129]
[5.66942609 3.86794331]
[5.7095317  3.87908242]
[5.75268041 3.90827223]
[5.79870789 3.91650832]
[5.84523533 3.94224692]
[5.89012244 3.949122  ]
[5.93386407 3.98709028]
[5.97711076 3.9918969 ]
[6.02094565 4.02558032]
[6.06694074 4.02284611]
[6.10963582 4.054326  ]
[6.15089281 4.06129808]
[6.19330884 4.09452594]
[6.23518943 4.08958975]
[6.27763623 4.12975364]
[6.31778237 4.12422303]
[6.35813768 4.16630167]
[6.39143879 4.15193754]
[6.42502445 4.17623454]
[6.46332772 4.17849416]
[6.50174023 4.20846034]
[6.53815205 4.20187055]
[6.57346094 4.22304649]
[6.60991746 4.24033014]
[6.64494132 4.25124678]
[6.67950596 4.26433764]
[6.71664479 4.27693843]
[6.75137383 4.28643394]
[6.78532004 4.30091335]
[6.82055518 4.3114841 ]
[6.8549548  4.32389528]
[6.89154713 4.33924024]
[6.92519753 4.34728002]
[6.95680143 4.36799192]
[6.9839873  4.36558193]
[7.01386296 4.37752926]
[7.04669408 4.38444213]
[7.08010141 4.4027094 ]
```

[7.11028822 4.39960745]
[7.14122798 4.42276966]
[7.17120807 4.42914397]
[7.20060102 4.44243605]
[7.2305107 4.44592803]
[7.26265944 4.45625171]
[7.29250806 4.46361034]
[7.32180853 4.47600726]
[7.35230215 4.48466821]
[7.38246108 4.49317433]
[7.41145318 4.50090903]
[7.43950554 4.5169756]
[7.46337706 4.52284216]
[7.485567   4.52180441]
[7.51098508 4.54024789]
[7.53960596 4.53836757]
[7.56874855 4.55178601]
[7.59638567 4.54941892]
[7.62312668 4.57549437]
[7.64948224 4.57142776]
[7.67654573 4.59357535]
[7.70591277 4.58229933]
[7.73225935 4.60365129]
[7.75732592 4.60188393]
[7.78368254 4.62481698]
[7.80965104 4.61162942]
[7.83639998 4.64182667]
[7.86100402 4.62812706]
[7.88579751 4.65970935]
[7.9039603 4.63752895]
[7.92242973 4.65226796]
[7.94570754 4.64782939]
[7.96941403 4.66867155]
[7.9912583 4.6547717]
[8.01212399 4.66755513]
[8.03425946 4.67777802]
[8.05515166 4.68112558]
[8.07568833 4.68646183]
[8.09892776 4.69228511]
[8.11993166 4.69454985]
[8.14029201 4.70248306]
[8.16207324 4.70549795]
[8.18313457 4.71157581]
[8.20653574 4.71982446]
[8.22711069 4.72135224]
[8.24566975 4.73430424]
[8.26003193 4.72648841]
[8.27707788 4.73098372]
[8.29711416 4.7334527]
[8.31788737 4.74406457]

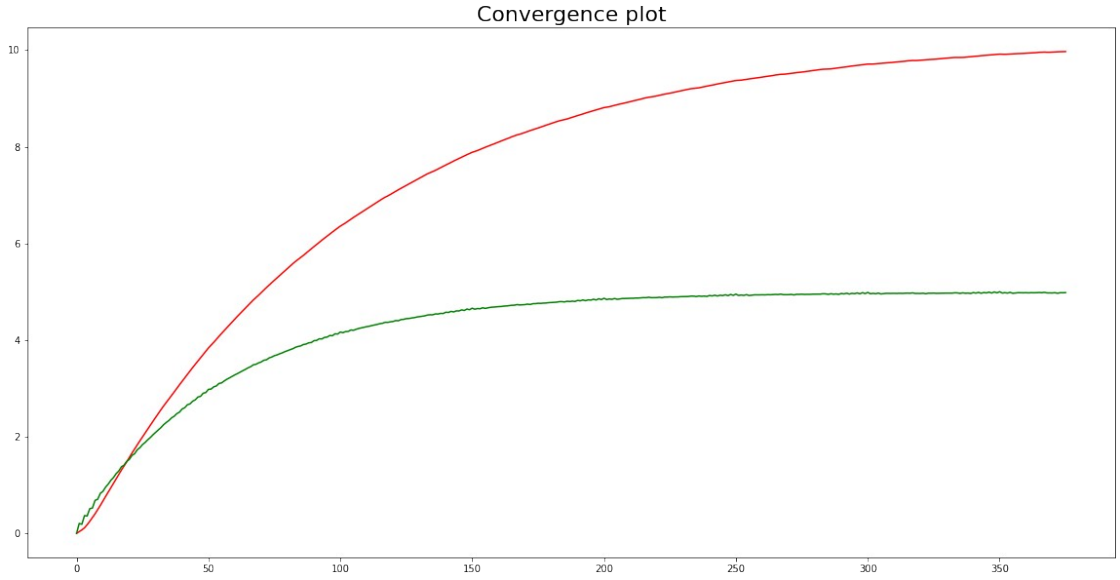
```
[8.33566293 4.73601713]
[8.35439526 4.75244019]
[8.37224409 4.75411202]
[8.38964713 4.761264  ]
[8.40768671 4.75960613]
[8.42810717 4.76457413]
[8.44636212 4.7669258 ]
[8.4641777  4.7739858 ]
[8.48329598 4.77763006]
[8.5022009  4.78108083]
[8.52004745 4.78388471]
[8.53701861 4.79435236]
[8.54998505 4.79543352]
[8.56133913 4.789119  ]
[8.57603634 4.80323564]
[8.59412287 4.7972551 ]
[8.61275519 4.80532036]
[8.62995839 4.79930484]
[8.64640718 4.82025331]
[8.66253605 4.81266757]
[8.67944895 4.82981919]
[8.69875022 4.81516937]
[8.71519007 4.83225894]
[8.7304428  4.8269573 ]
[8.74706881 4.84548506]
[8.76339562 4.82900245]
[8.78062842 4.85494626]
[8.79581239 4.83792965]
[8.8111944  4.8648813 ]
[8.82020048 4.83953554]
[8.82953558 4.85012568]
[8.84372652 4.84317373]
[8.85852821 4.86006536]
[8.87155175 4.8432295 ]
[8.8836711  4.85243805]
[8.89713194 4.85984661]
[8.90945986 4.86003467]
[8.92149903 4.86210789]
[8.93631629 4.86518732]
[8.948999   4.8644478 ]
[8.96112019 4.86972592]
[8.9747423  4.86953621]
[8.98771644 4.8730306 ]
[9.00311738 4.87830745]
[9.01576446 4.8770999 ]
[9.02643676 4.88662991]
[9.03304377 4.87670995]
[9.04233728 4.87790007]
[9.05463394 4.87883026]
[9.06776246 4.88596904]
```

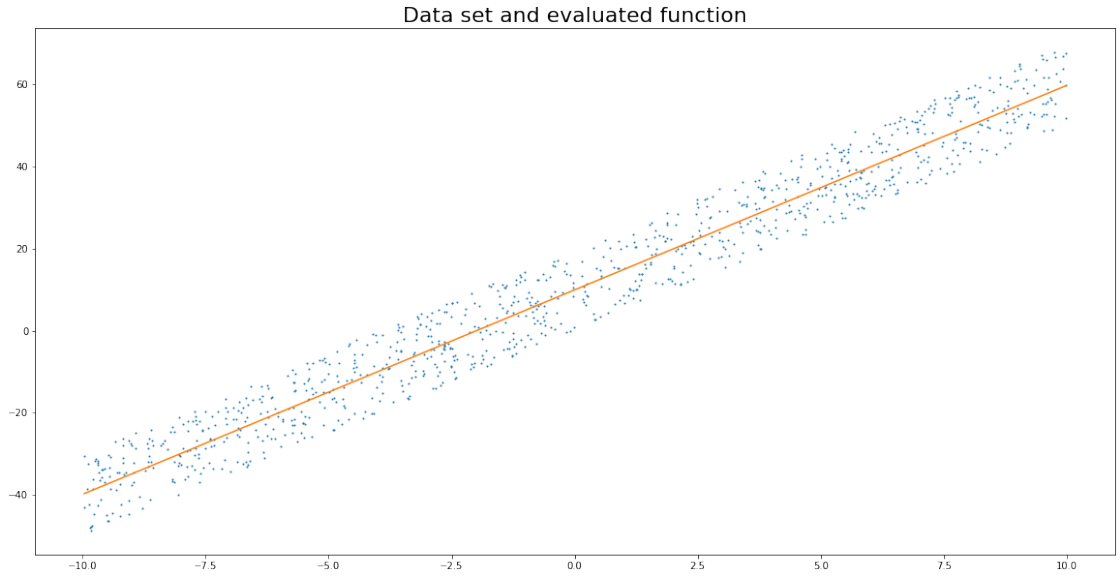
[9.07802837 4.87603507]
[9.08936851 4.88951828]
[9.09986958 4.88939884]
[9.11000886 4.89390794]
[9.12085797 4.89018466]
[9.1341718 4.8929225]
[9.14539695 4.89325895]
[9.15624953 4.8980666]
[9.16847031 4.89968152]
[9.18055184 4.90100863]
[9.19164004 4.90179794]
[9.20189535 4.90978547]
[9.20826216 4.90891201]
[9.21305834 4.90027452]
[9.22126853 4.91267851]
[9.23297698 4.90505803]
[9.24525034 4.91075844]
[9.25613863 4.90334432]
[9.26635446 4.92205138]
[9.27628973 4.91310616]
[9.28705663 4.92806025]
[9.30026244 4.91211594]
[9.3106989 4.92740275]
[9.32000333 4.92069244]
[9.33073306 4.93731345]
[9.34121734 4.9195436]
[9.35268191 4.94365315]
[9.36215656 4.92530609]
[9.37184302 4.95017148]
[9.37530716 4.92356132]
[9.37911761 4.93232101]
[9.38780955 4.92448916]
[9.39721776 4.93964361]
[9.40489884 4.92165014]
[9.41172041 4.92932103]
[9.41992597 4.9356335]
[9.42706364 4.93450082]
[9.43395465 4.93519427]
[9.44366836 4.9371782]
[9.45130677 4.93519133]
[9.45843239 4.93940635]
[9.46710749 4.93784647]
[9.47517895 4.94029729]
[9.48572901 4.94432603]
[9.49356997 4.94196476]
[9.49947049 4.94995386]
[9.50138443 4.9392464]
[9.50598919 4.93893735]
[9.51360139 4.9394153]
[9.52210197 4.94493458]

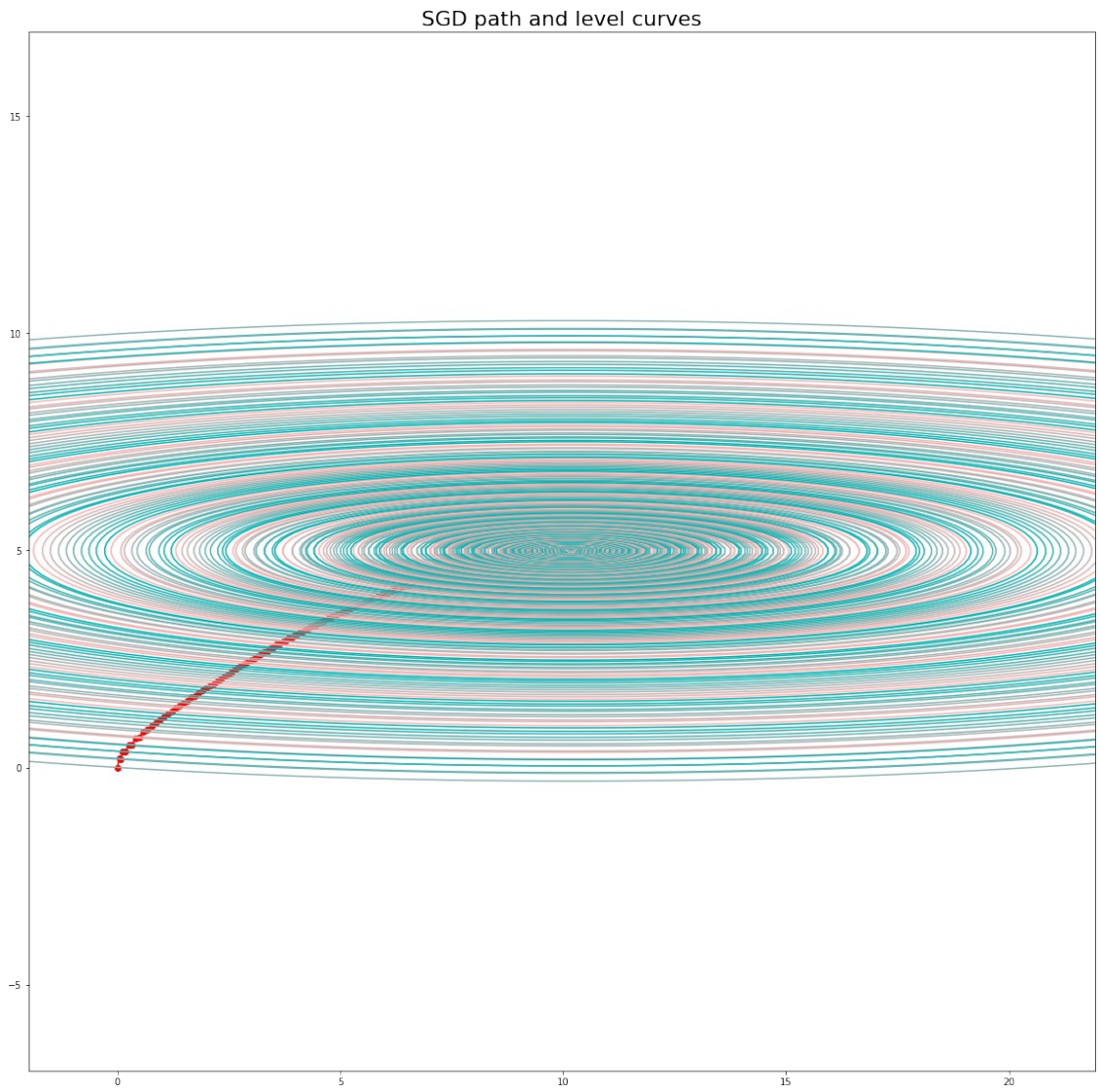
[9.52782177 4.93431945]
[9.53468445 4.94649715]
[9.54073427 4.94573263]
[9.54647253 4.9490887]
[9.55296531 4.94455294]
[9.56197274 4.94636178]
[9.56893607 4.94590103]
[9.57556767 4.9497523]
[9.58360683 4.95055803]
[9.59155185 4.95098927]
[9.5985427 4.95096284]
[9.60472786 4.95782723]
[9.60709891 4.95616262]
[9.60792453 4.9464814]
[9.6122076 4.95822594]
[9.62005325 4.94996316]
[9.6284773 4.9545963]
[9.63554203 4.94667265]
[9.64198233 4.96438157]
[9.64816578 4.95493421]
[9.65521032 4.96890196]
[9.66472406 4.95248528]
[9.67152243 4.96700428]
[9.67722166 4.95974537]
[9.68437837 4.97552831]
[9.69132216 4.95727365]
[9.69929052 4.98058091]
[9.70530495 4.96171041]
[9.71154309 4.98561108]
[9.71165167 4.95850676]
[9.71211861 4.9664417]
[9.71748126 4.95834426]
[9.72362215 4.9727193]
[9.72806685 4.95428297]
[9.73167894 4.96128365]
[9.73670039 4.96718545]
[9.74069259 4.96549954]
[9.74446433 4.96560552]
[9.75108534 4.96716163]
[9.75566615 4.96465738]
[9.75976331 4.9684553]
[9.76543938 4.96630305]
[9.770539   4.96833867]
[9.77814824 4.97184108]
[9.78307589 4.96899031]
[9.7860878 4.97626733]
[9.78516043 4.96528553]
[9.78692759 4.96427761]
[9.79170337 4.96468135]
[9.79740134 4.96942109]

```
[9.80036812 4.95858545]
[9.80451832 4.97017008]
[9.80787126 4.96919791]
[9.81094278 4.9720419 ]
[9.81479604 4.9671965 ]
[9.82119383 4.96861818]
[9.82557373 4.96785074]
[9.82964679 4.97129245]
[9.83515111 4.97178281]
[9.84058873 4.97183389]
[9.84509579 4.97148168]
[9.84881426 4.97782348]
[9.84876534 4.97584311]
[9.84718622 4.96568083]
[9.84909109 4.97718321]
[9.8545968  4.96867747]
[9.86068988 4.97281606]
[9.86543885 4.96472428]
[9.86959183 4.98197818]
[9.87350226 4.97236039]
[9.87829189 4.98587413]
[9.88556893 4.96930106]
[9.89016257 4.98348985]
[9.89367679 4.97602541]
[9.89866827 4.99143257]
[9.90346651 4.97300697]
[9.90931586 4.99595674]
[9.9132332  4.97688671]
[9.91738295 5.0003274 ]
[9.91545921 4.97303682]
[9.91390179 4.98059035]
[9.91724815 4.97244332]
[9.92140947 4.98645977]
[9.92389337 4.9678627 ]
[9.9255609  4.97456609]
[9.92865298 4.98032452]
[9.93073892 4.97840619]
[9.93262054 4.97825964]
[9.93736737 4.97965471]
[9.94009494 4.97693618]
[9.94235634 4.98057585]
[9.94621449 4.97816362]
[9.94951281 4.98003801]
[9.95533938 4.98331719]
[9.9585012  4.98025702]
[9.95976387 4.98719476]
[9.95711576 4.97613113]
[9.95716488 4.97478691]
[9.96022272 4.97522885]
[9.96422308 4.97958019]
```

```
[9.9655223  4.96869093]
[9.96802907 4.97999858]
[9.96974792 4.97897683]
[9.97120348 4.98158815]]
```





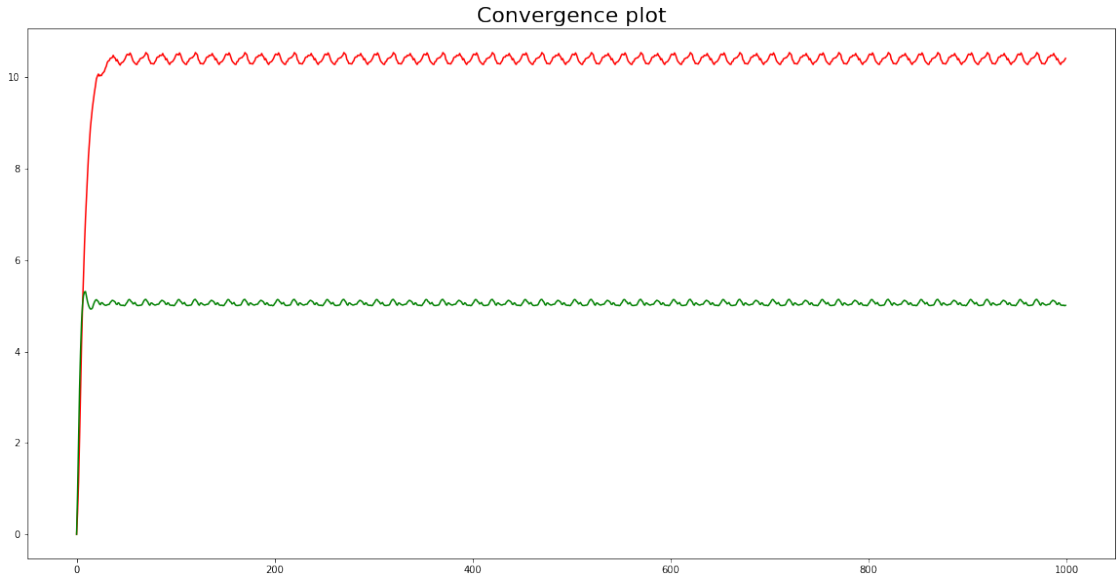


Нагрузочное тестирование SGD_Momentum

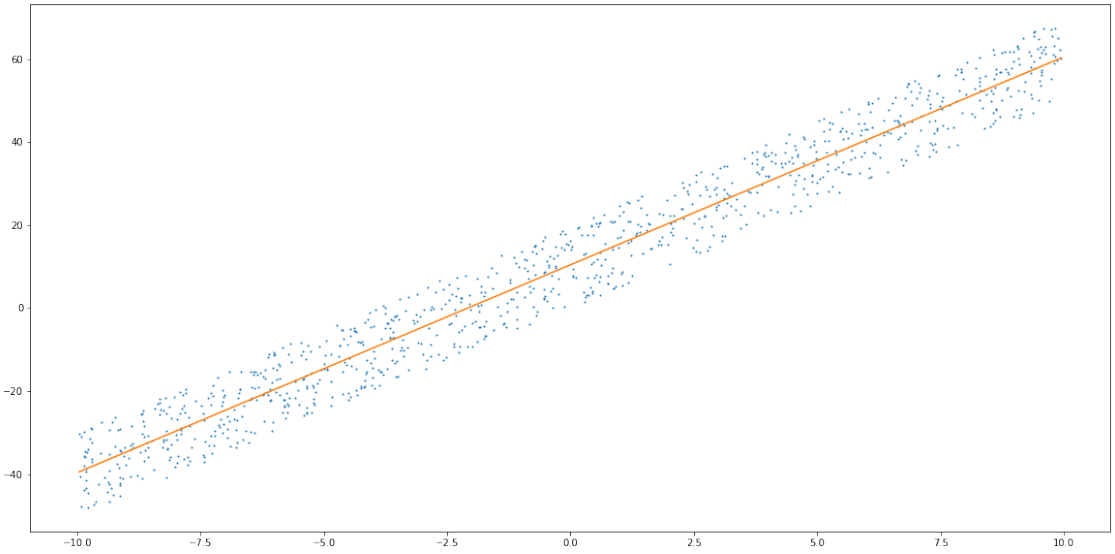
```
a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
infinity_epoch = 1000
batch_size = 60
lr = [60, 5]
b = [0.5]
"""При попытке повысить точность momentum "перескакивает" через точку минимума"""
stop_criteria:Callable[[List[float]], bool]=lambda x: np.sum(np.abs(x - a)) < 0.05
```

```
start_time = time.time()
points = sgd_momentum(sum_fun, x, infinity_epoch, batch_size, lr, b,
stop_criteria=stop_criteria)
print(f'Process time: {(time.time() - start_time) / len(points)}')

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

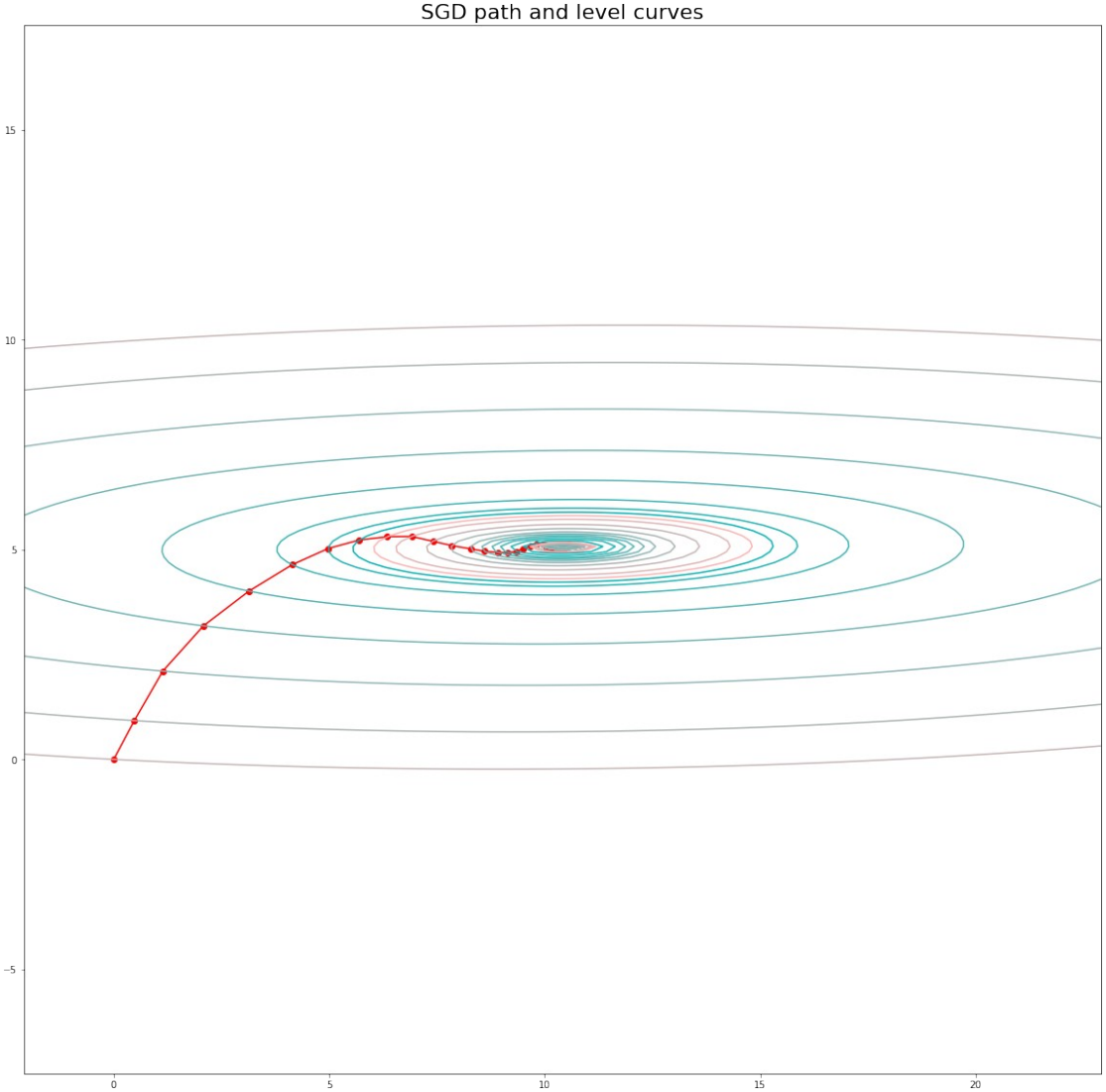
```
Process time: 0.004205939769744873
Precision: [-0.41515528 -0.01061543]
Min point: [10.41515528  5.01061543]
Iterations: 1000
Path: [[ 0.          0.        ]
 [ 0.47789999  0.92365833]
 [ 1.13910246  2.10389513]
 ...
 [10.34660472  5.01560359]
 [10.37856238  5.00216782]
 [10.41515528  5.01061543]]
```



Data set and evaluated function



SGD path and level curves



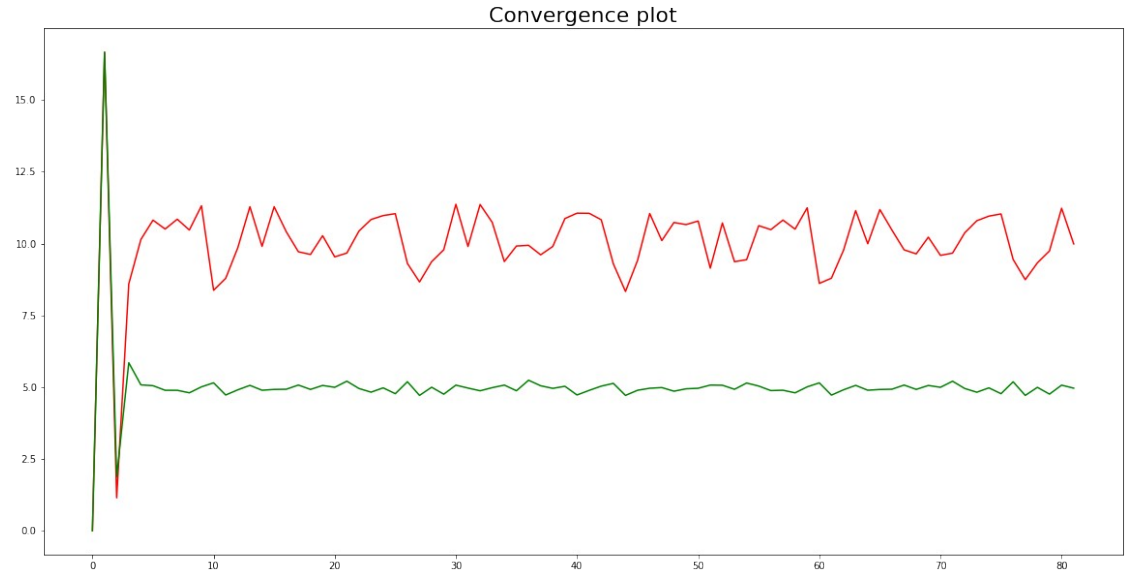
Нагрузочное тестирование SGD_AdaGrad

```
a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
infinity_epoch = 1000
batch_size = 60
lr = 1000
"""
Удалось повысить точность в 5 раз, количество итераций как у
Нестерова, но понижение в 10 раз влекло те же последствия, что и
Momentum:
по второй координате заваливались по другую сторону числовой оси
upd: повышение точности оказалось ненадежным: успешен 1 прогон из 5
"""
stop_criteria:Callable[[List[float]], bool]=lambda x: np.sum(np.abs(x
- a)) < 0.05

start_time = time.time()
points = sgd_adagrad(sum_fun, x, infinity_epoch, batch_size, lr,
stop_criteria=stop_criteria)
print(f'Process time: {(time.time() - start_time) / len(points)}')

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

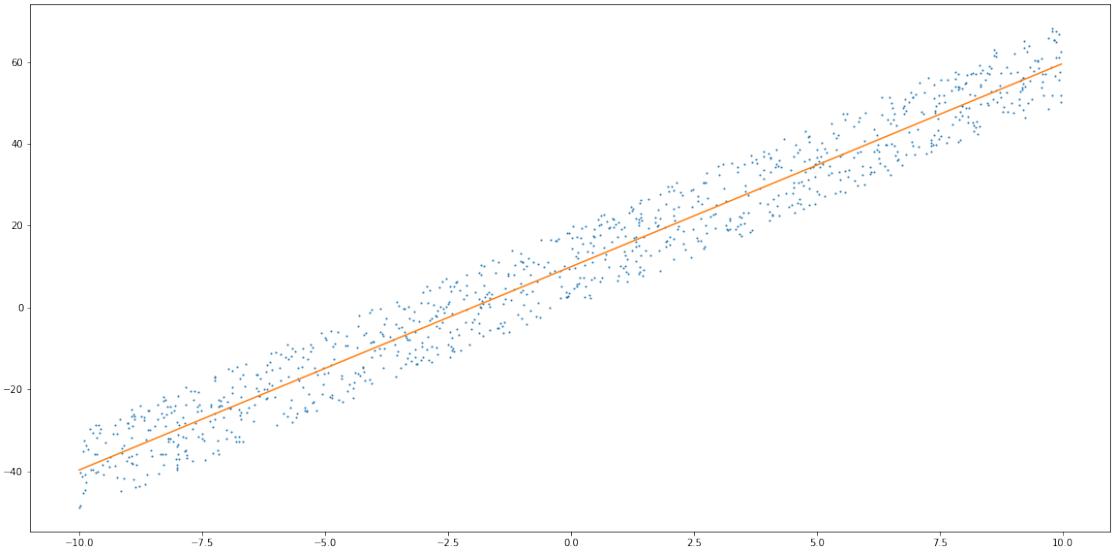
```
Process time: 0.0041847083626723874
Precision: [0.01220052 0.03707272]
Min point: [9.98779948 4.96292728]
Iterations: 82
Path: [[ 0.          0.        ]
 [16.66666619 16.66666665]
 [ 1.14327082  1.87612025]
 [ 8.59503484  5.8494939 ]
 [10.14305088  5.07830889]
 [10.81640694  5.04974473]
 [10.5064436   4.88965238]
 [10.8483761   4.88922693]
 [10.47273238  4.80179133]
 [11.31550342  5.00915037]
 [ 8.37288633  5.14989928]
 [ 8.78811943  4.7249312 ]
 [ 9.8561322   4.90605026]
 [11.28404288  5.06098358]
 [ 9.90228416  4.89113951]
 [11.28323129  4.91936712]
```

```
[10.41214235  4.9264034 ]
[ 9.71411857  5.07182598]
[ 9.62088694  4.91954466]
[10.27376395  5.05863434]
[ 9.53709313  4.99279252]
[ 9.6694129   5.20634749]
[10.43492951  4.95141109]
[10.83833098  4.82453431]
[10.97546555  4.97400299]
[11.04133015  4.77081347]
[ 9.30773908  5.1873124 ]
[ 8.66305936  4.71103095]
[ 9.36999692  4.99642404]
[ 9.78438374  4.75342007]
[11.36838294  5.06891343]
[ 9.89769767  4.96663116]
[11.36088645  4.87248175]
[10.73936603  4.98061387]
[ 9.37368192  5.07027487]
[ 9.91509372  4.87372776]
[ 9.93886062  5.2394043 ]
[ 9.60846275  5.04704328]
[ 9.89956427  4.95618472]
[10.87091714  5.0290843 ]
[11.05924876  4.72577985]
[11.05497561  4.88604692]
[10.8294962   5.03382062]
[ 9.29269998  5.13076894]
[ 8.33179107  4.71106388]
[ 9.41652882  4.89050698]
[11.04732827  4.96026988]
[10.10361636  4.98409925]
[10.73478082  4.85757749]
[10.66086571  4.9411813 ]
[10.78587931  4.96002227]
[ 9.14548922  5.0707939 ]
[10.71769162  5.06523954]
[ 9.36807008  4.92427311]
[ 9.43999916  5.14235138]
[10.62376223  5.0369273 ]
[10.48539919  4.87901193]
[10.81849818  4.89155916]
[10.50259335  4.80097327]
[11.24563765  5.00880462]
[ 8.61229797  5.14834104]
[ 8.79434128  4.72156475]
[ 9.76944145  4.90662653]
[11.14792117  5.06063713]
[ 9.99390856  4.89186642]
[11.18487761  4.91766867]
```

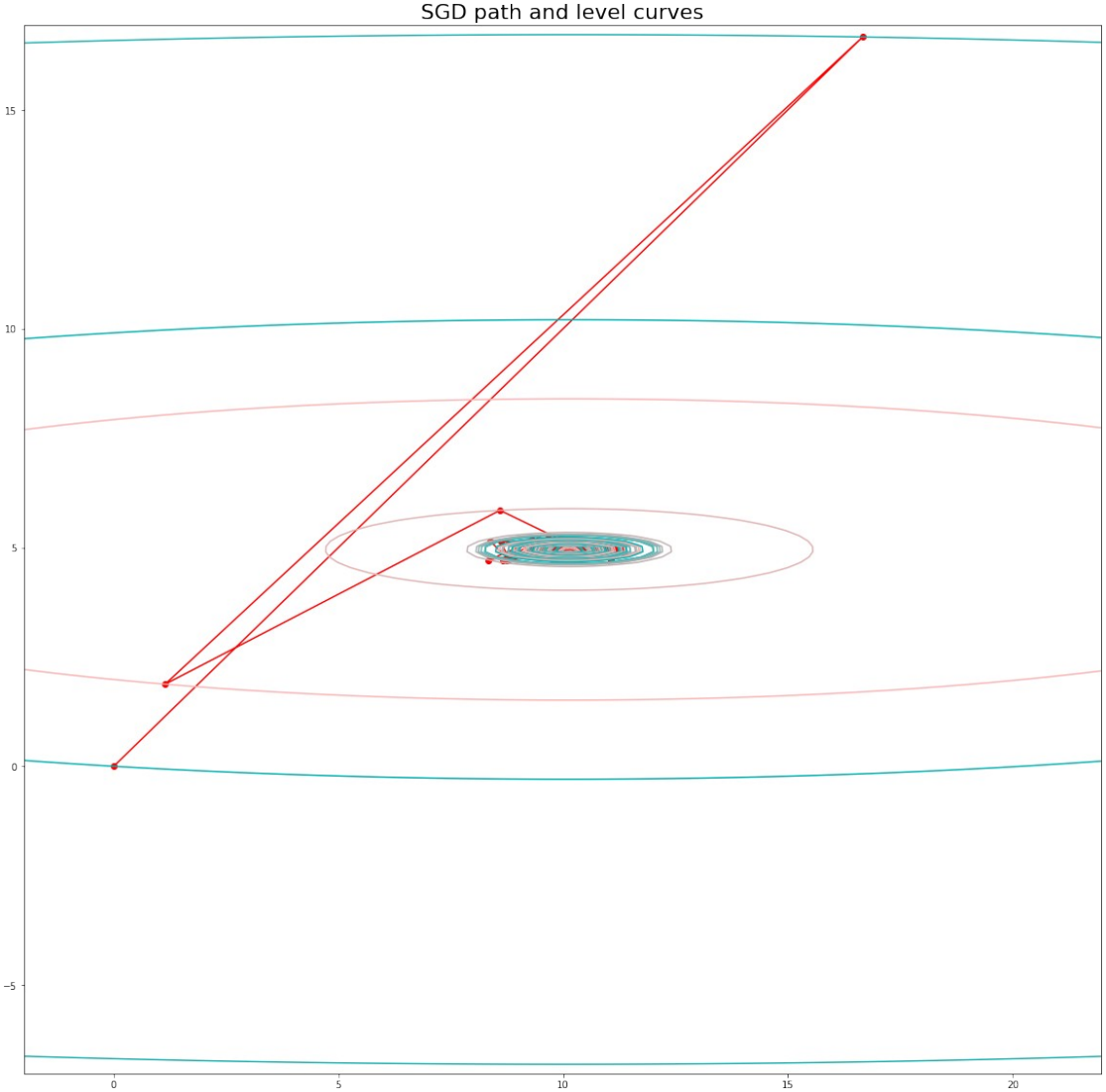
```
[10.46647459  4.92737896]
[ 9.7815995   5.07142208]
[ 9.64074866  4.92024099]
[10.22335378  5.05762792]
[ 9.5882374   4.99281623]
[ 9.6678293   5.20564341]
[10.37199927  4.95253751]
[10.79500086  4.82181289]
[10.95612531  4.97379784]
[11.03228524  4.77108674]
[ 9.44700295  5.18542924]
[ 8.74226985  4.71114665]
[ 9.32744937  4.99231065]
[ 9.74327561  4.7566136 ]
[11.23086933  5.06847611]
[ 9.98779948  4.96292728]]
```



Data set and evaluated function



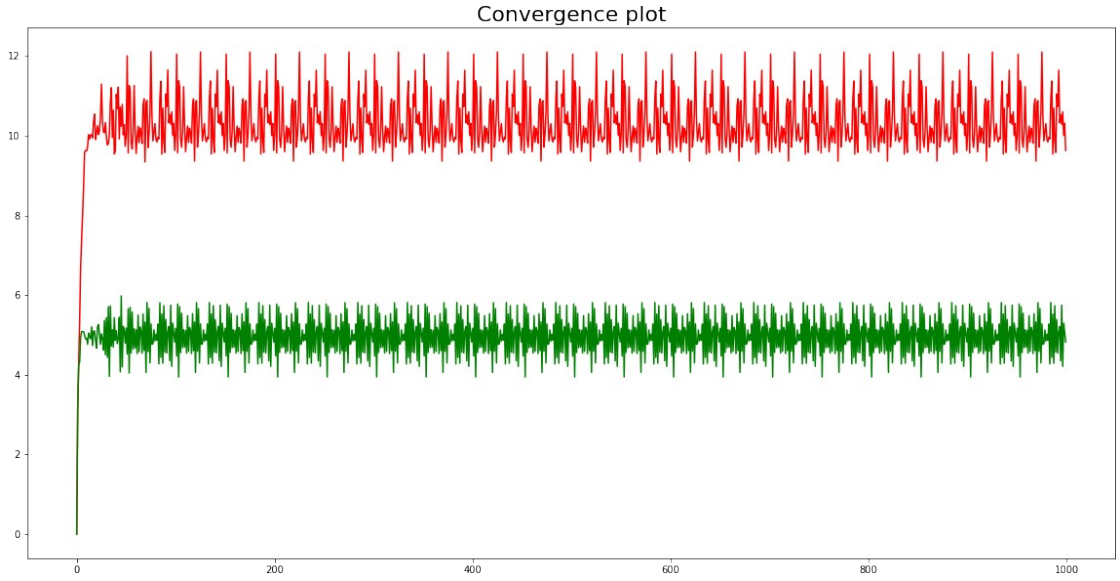
SGD path and level curves

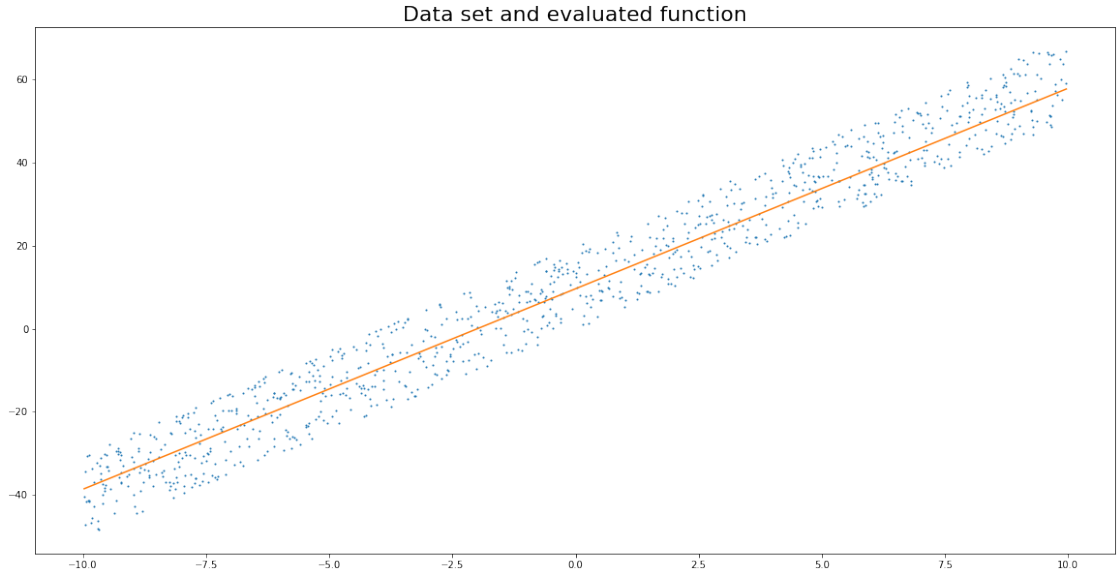


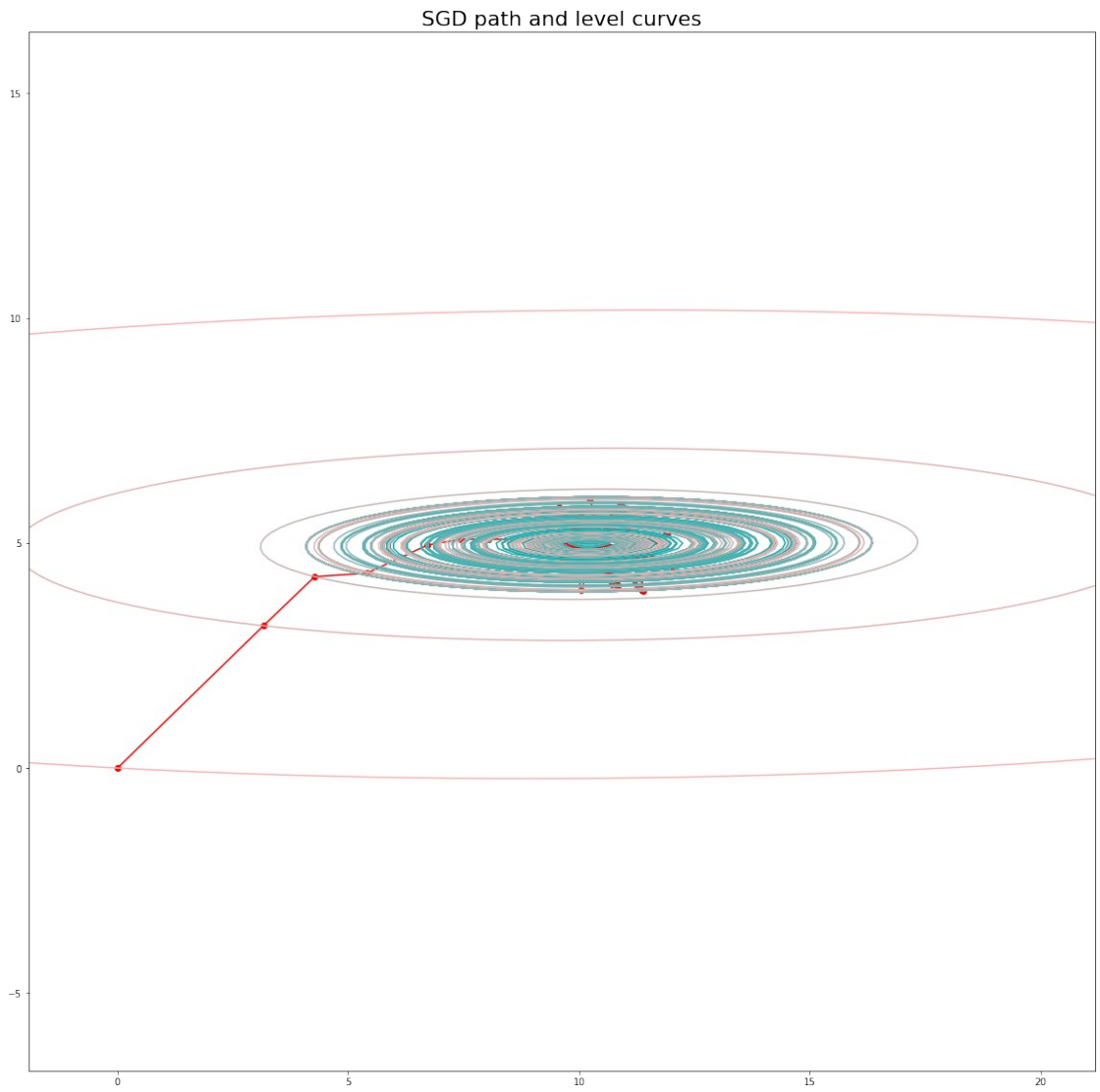
Нагрузочное тестирование SGD_RMSProp

```
a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
infinity_epoch = 1000
batch_size = 60
lr = 60
b = 0.9
"""Попытки повысить точность не увенчались успехом"""
stop_criteria:Callable[[List[float]], bool]=lambda x: np.sum(np.abs(x
- a)) < 0.05

start_time = time.time()
points = sgd_rmsprop(sum_fun, x, infinity_epoch, batch_size, lr, b,
stop_criteria=stop_criteria)
print(f'Process time: {(time.time() - start_time) / len(points)}')

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Process time: 0.004076819896697998
Precision: [0.37097417 0.17680931]
Min point: [9.62902583 4.82319069]
Iterations: 1000
Path: [[ 0.          0.        ]
 [ 3.16227752  3.16227765]
 [ 4.27241067  4.25357438]
 ...
 [10.00423246  5.29686053]
 [10.30880668  5.04801048]
 [ 9.62902583  4.82319069]]
```


Convergence plot
12
10
8
6
4
2
0
0
200
400
600
800
1000


Data set and evaluated function
60
40
20
0
−20
−40
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



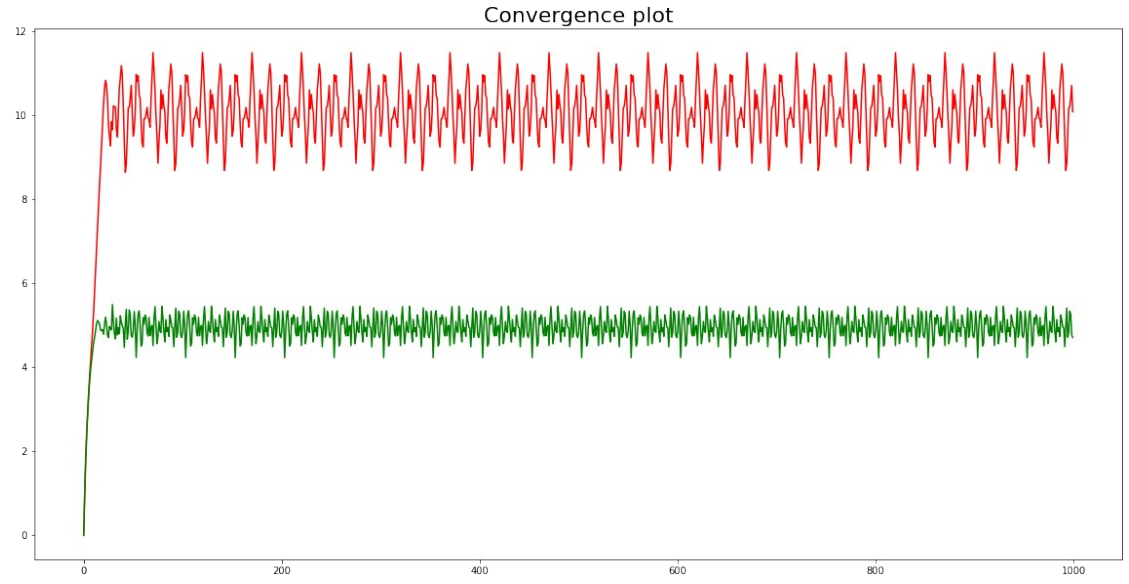
Нагрузочное тестирование SGD_Adam

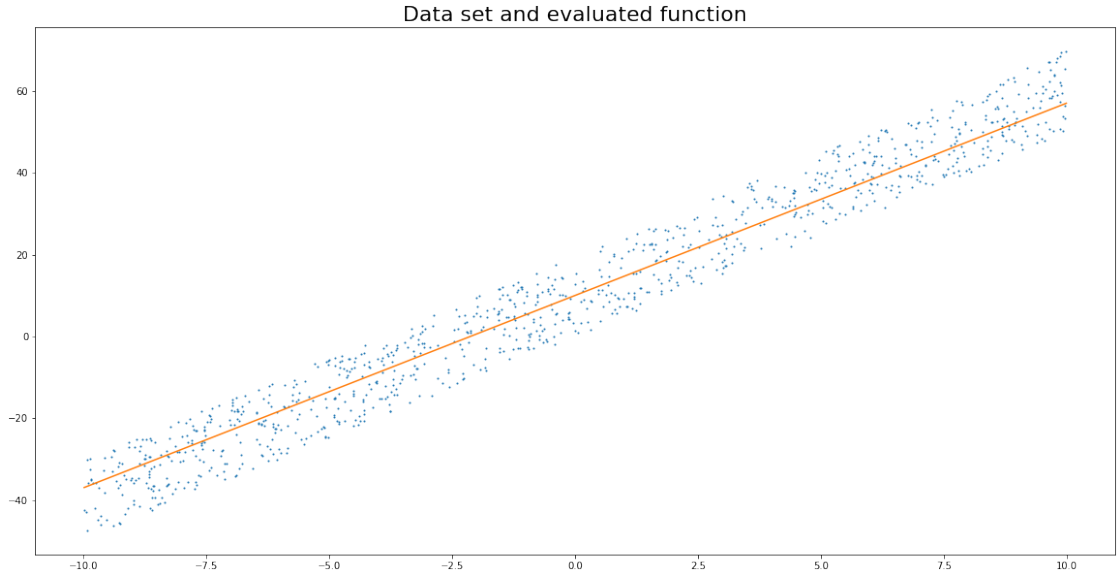
```
a = [10, 5]
t, ft = generate_dataset(a, 10, 1000, (-10, 10))
sum_fun = generate_minimized_two_variable_fun(t, ft)

# SGD params
x = np.zeros(2)
infinity_epoch = 1000
batch_size = 60
lr = 60
b1 = 0.5
b2 = 0.7
scheduler = lambda lr: lr * np.exp(-0.05)
"""
```

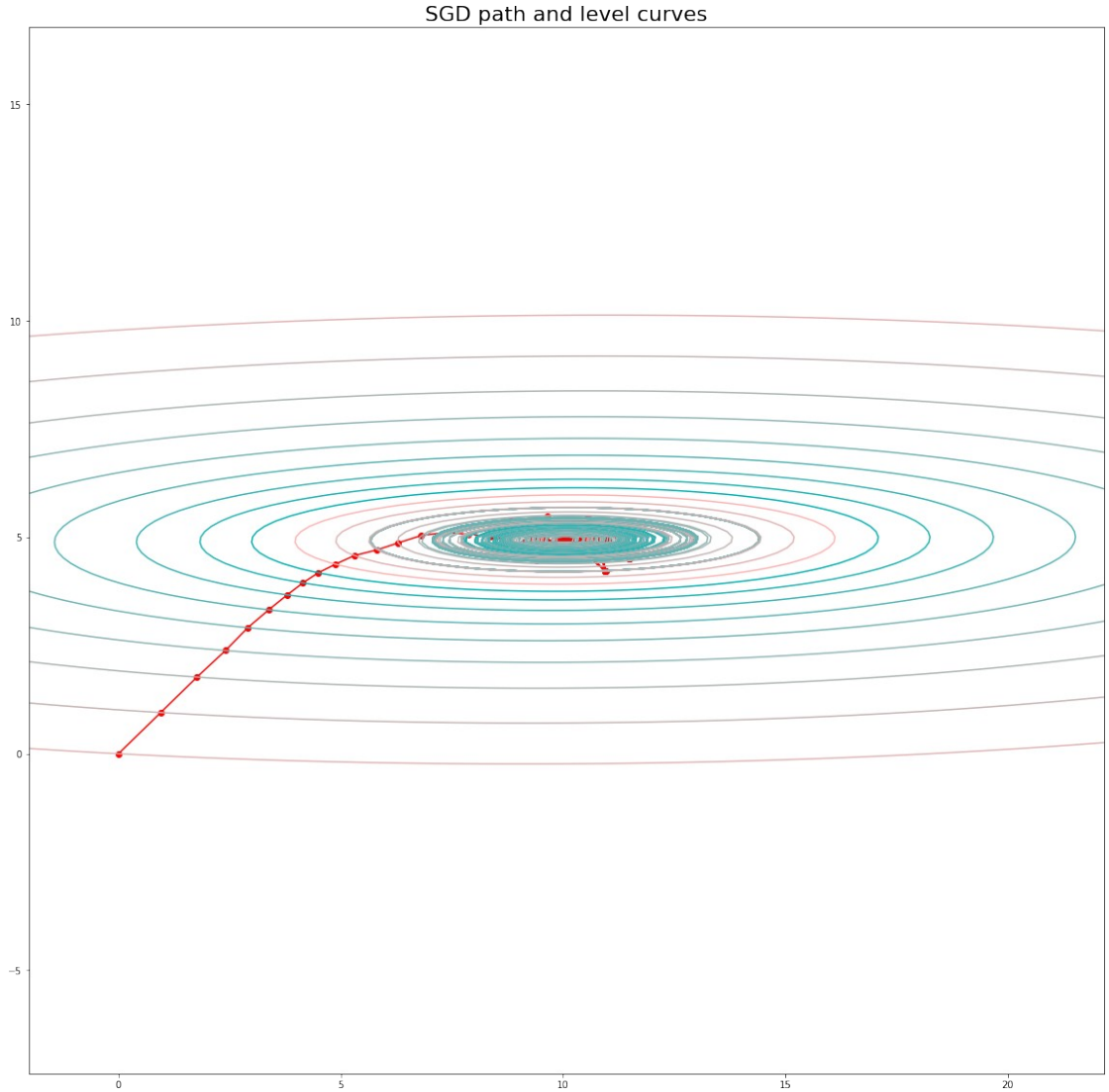
Повысить точность так же не удалось, но Adam показывает наилучшую надежность: 7 из 10 запусков были удачны и достигли

```
желаемой точности за 10-200 итераций
"""
stop_criteria:Callable[[List[float]], bool]=lambda x: np.sum(np.abs(x
- a)) < 0.05

start_time = time.time()
points = sgd_adam(sum_fun, x, infinity_epoch, batch_size, lr, b1, b2,
scheduler=scheduler, stop_criteria=stop_criteria)
"""
0.03123021125793457
1.8599903583526611
"""
print(f'Process time: {(time.time() - start_time) / len(points)}')

print_result(a, points)
plot_convergence(points)
plot_dataset_and_function(t, ft, points[-1])
plot_path_contours(sum_fun, points)
```

```
Process time: 0.004203010559082032
Precision: [-0.07805058  0.29669885]
Min point: [10.07805058  4.70330115]
Iterations: 1000
Path: [[ 0.          0.        ]
 [ 0.95122941  0.95122942]
 [ 1.76536341  1.76989631]
 ...
 [10.38833955  5.26213357]
 [10.70464863  4.76653419]
 [10.07805058  4.70330115]]
```




Data set and evaluated function
60
40
20
0
−20
−40
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


SGD path and level curves
15
10
5
0
−5
0
5
10
15
20

Результаты и анализ нагрузочных тестов

Время выполнения одной итерации алгоритмами:

Nesterov: 1.6084521788824318 ms Momentum: 1.6133334135627009 ms AdaGrad: 1.8321524513053467 ms RMSProp: 1.8016457349168367 ms Adam: 1.8647285017406509 ms

Использование памяти:

Nesterov, Momentum, AdaGrad, RMSProp хранят одно дополнительное значение. Adam хранит два дополнительных значения. Созданные реализации сохраняют данную корреляцию (все дополнительыне вычисляемые значения обернуты в листы из одного элемента). Таким образом, все алгоритмы требуют больше RAM, чем обычный градиентный спуск, при этом Adam расходует больше всего оперативной памяти.

Арифметические операции:

Nesterov и Momentum требуют дополнительную операцию умножения на дополнительное значение (b). AdaGrad возводит дополнительное значение в квадрат и делит.RMSProp возводит в квадрат и умножает. Adam включает в себя Momentum и RMSProp, и поэтому требует больше всего арифметических операций.

Надежность

Ни один из алгоритмов не обладает абсолютной точностью. Больше всего выделяется алгоритм Нестерова: примерно 1 из 10 запусков заканчивается сходимостью). В среднем, алгоритмы успешно выполняют работу в 1 из 5 наборов сгенерированных датасетов

Скорость сходимости

Худший результат: Nesterov: 500-700 итераций до достижения заявленной погрешности. Остальные алгоритмы могут показать результат: < 100 итераций. В некоторых случаях Momentum, AdaGrad, RMSProp сходятся за 300-500 итераций. Выделяетсся алгоритм Adam: число итераций до достижения погрешности с этим алгоритмом стабильно в пределах от 10 до 200.

Вывод

Наименее предпочтительный алгоритм - Nesterov, так как не отличается ни надежностью, ни скоростью при потреблении ресурсов (количество арифметических операций и затраченной RAM) сопоставим с Momentum, AdaGrad и RMSProp. Наилучший результат показывает Adam, так как совмещает в себе два алгоритма Momentum и RMSProp.Увеличение

производительности и надежности в сравнении с увеличением затрачиваемых ресурсов выгодно на фоне остальных алгоритмов.