Одиночный нейрон

С.И.Хашин
http://math.ivanovo.ac.ru/dalgebra/Khashin/index.html

Ивановский государственный университет

.

Иваново-2019

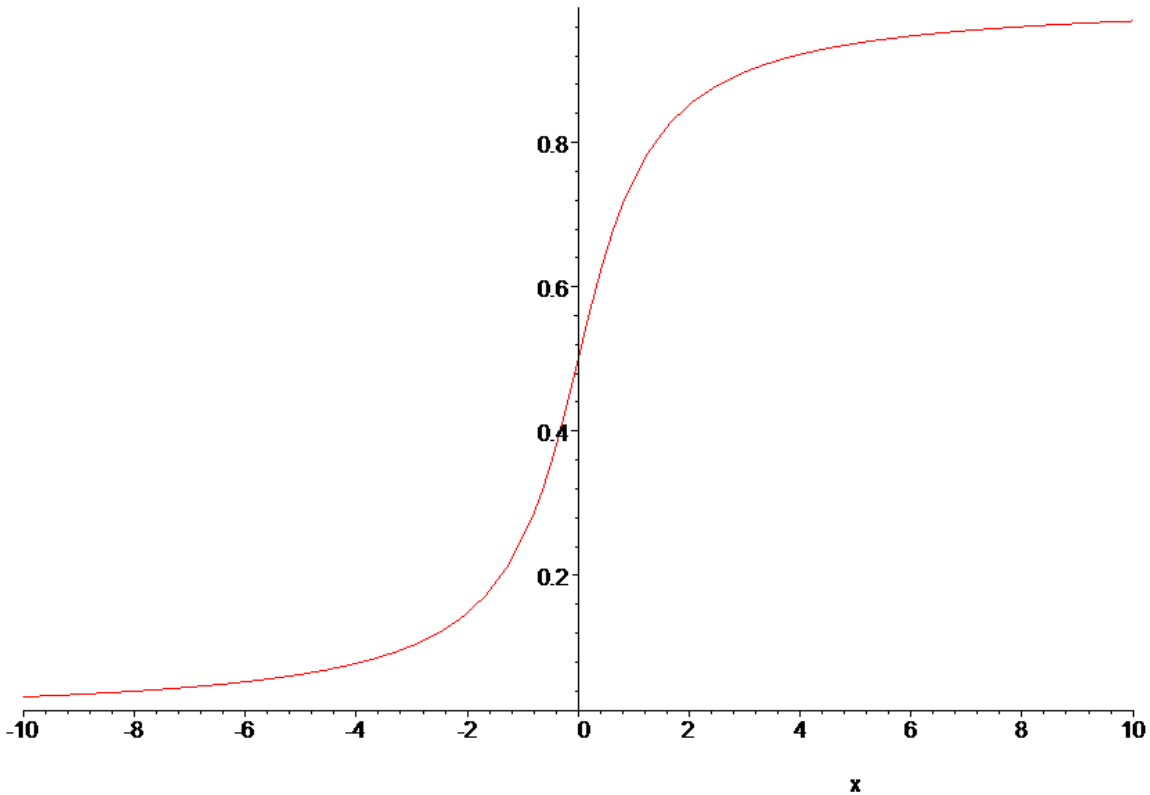
План

Нейрон

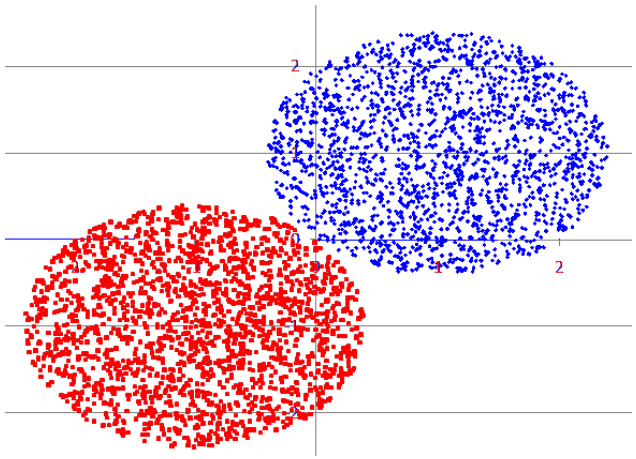
Определение 0. *Нейроном* $W = (f, \{w_0, \ldots, w_n\})$ будем называть функцию переменных $(x_1, \ldots, x_n)$ вида

$$W(x_1, \ldots, x_n) = f(w_0 + w_1 x_1 + \cdots + w_n x_n),$$

где w_i – внутренние параметры нейрона (весовые коэффициенты) и f – (передаточная) функция.



Обучение одного нейрона



$$L = w_0 + w_1 x_1 + \cdots + w_n x_n.$$

На A : $L > 0$, на B : $L < 0$.

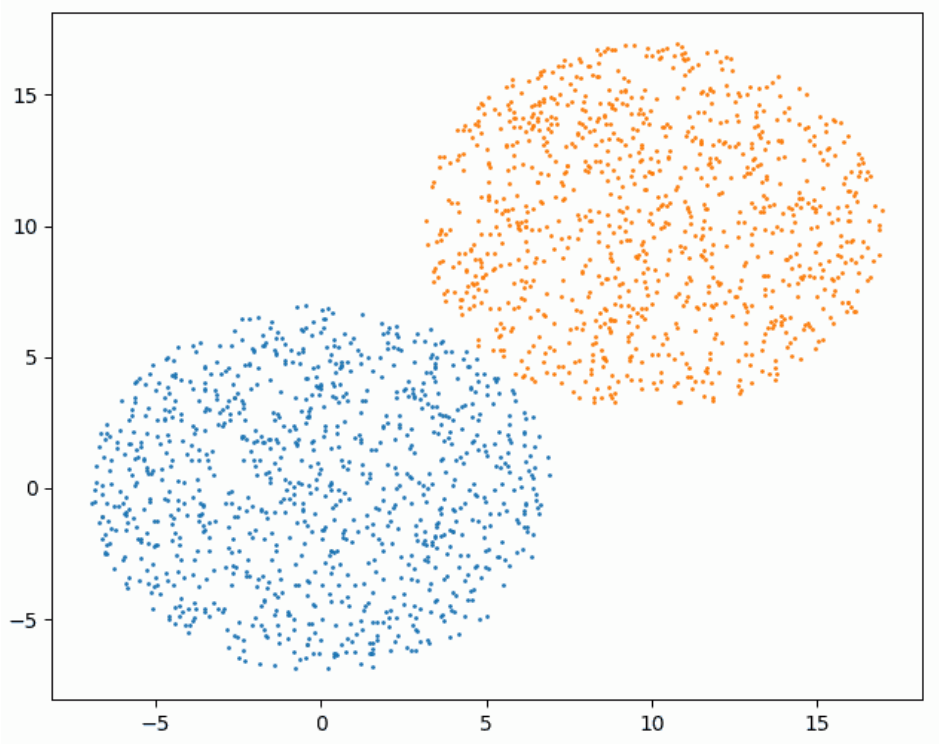
Коррекция-A: $w_0 + +$; $w_i + = x_i$;

Коррекция-B: $w_0 - -$; $w_i - = x_i$;

Теорема Новикова: если есть решение, то обучение закончится за конечное время.

Обучение одного нейрона (neuron_AB.csv)

```
A: 0.00 0.00 7.00, B:10.0010.00 7.00; 1000 points
A,  -4.8627,  -2.7670
B,  15.1039,   6.7192
A,  -6.1315,  -0.5620
B,   8.7361,   9.4905
...
```



Обучение одного нейрона

```
def load_data_01(fname):
    ''' читаем обучающие данные из fname, return Y, X
    Y: A -> 0, B -> 1
    '''
     X = np.genfromtxt(fname, skip_header=1, delimiter=',',
         usecols=(1,2))
     Ys = np.genfromtxt(fname, skip_header=1, delimiter=',',
         usecols=(0), dtype=str)
     Y = np.zeros(len(X), dtype=int)
     Y[Ys=='B'] = 1
     return Y, X

Y,X = load_data_01('neuron_AB.csv')
print(X[:4])
print(Y[:4])
```

teach_01(Y,X)

```
def teach_01(Y,X):  # обучение, возвращаем w0, w
    w0, w=0, np.zeros(2)
    N = len(X)
    for i in range(10): # 10 эпох обучения
        n_corr = 0  # счётчик коррекций
        for j in range(N):
            result = w0 + np.dot(w,X[j])
            if Y[j]==0 and result <= 0:
                w0 += 1; w += X[j]
                n_corr += 1
            if Y[j] == 1 and result >= 0:
                w0 -= 1; w -= X[j]
                n_corr += 1
        print(f'epoch {i}, n_corr={n_corr}')
        if n_corr==0: break
    return w0, w
```

Ответ-1

```
Y,X = load_data_01('neuron_AB.csv')
w0, w = teach_01(Y,X)
print(f'w={w0:8.4f}', w)

>epoch 0, n_corr=37
>epoch 1, n_corr=0
>w= 23.0000 [-2.0623 -2.433 ]
```

neuron_AB_.csv

Скопируем файл neuron_AB.csv в neuron_AB_.csv и немного испортим:

```
A: 0.00 0.00 7.00, B:10.0010.00 7.00; 1000 points
B,  -4.8627,  -2.7670 было A
A,  15.1039,   6.7192 было B
A,  -6.1315,  -0.5620
B,   8.7361,   9.4905
A,   3.1778,   3.7589
...
```

Ответ-2

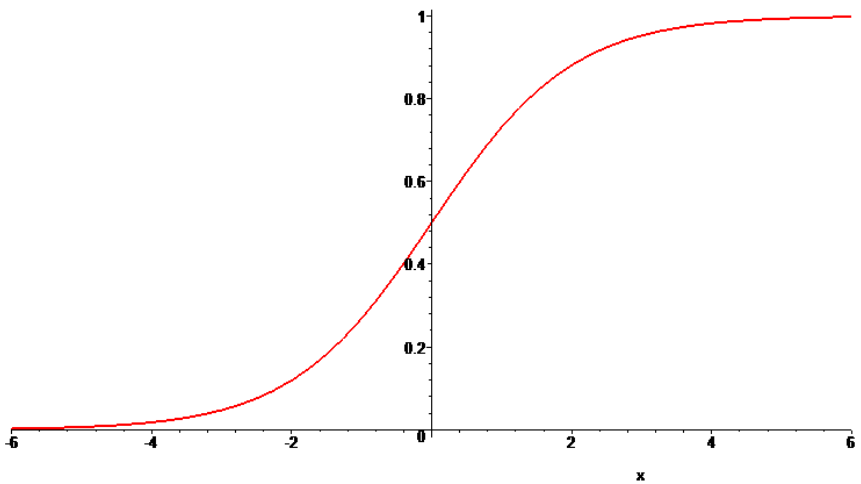
```
Y,X = load_data_01('neuron_AB_.csv')
w0, w = teach_01(Y,X)
print(f'w={w0:8.4f}', w)

>epoch 0, n_corr=83
>epoch 1, n_corr=48
>epoch 2, n_corr=35
>epoch 3, n_corr=54
>epoch 4, n_corr=47
>epoch 5, n_corr=35
>epoch 6, n_corr=11
>epoch 7, n_corr=20
>epoch 8, n_corr=37
>epoch 9, n_corr=37
>w=105.0000 [-11.8813  -8.6166]
```

Нейрон

Определение. *Нейроном* $W = (f, \{w_0, \ldots, w_n\})$ будем называть функцию переменных $(x_1, \ldots, x_n)$ вида

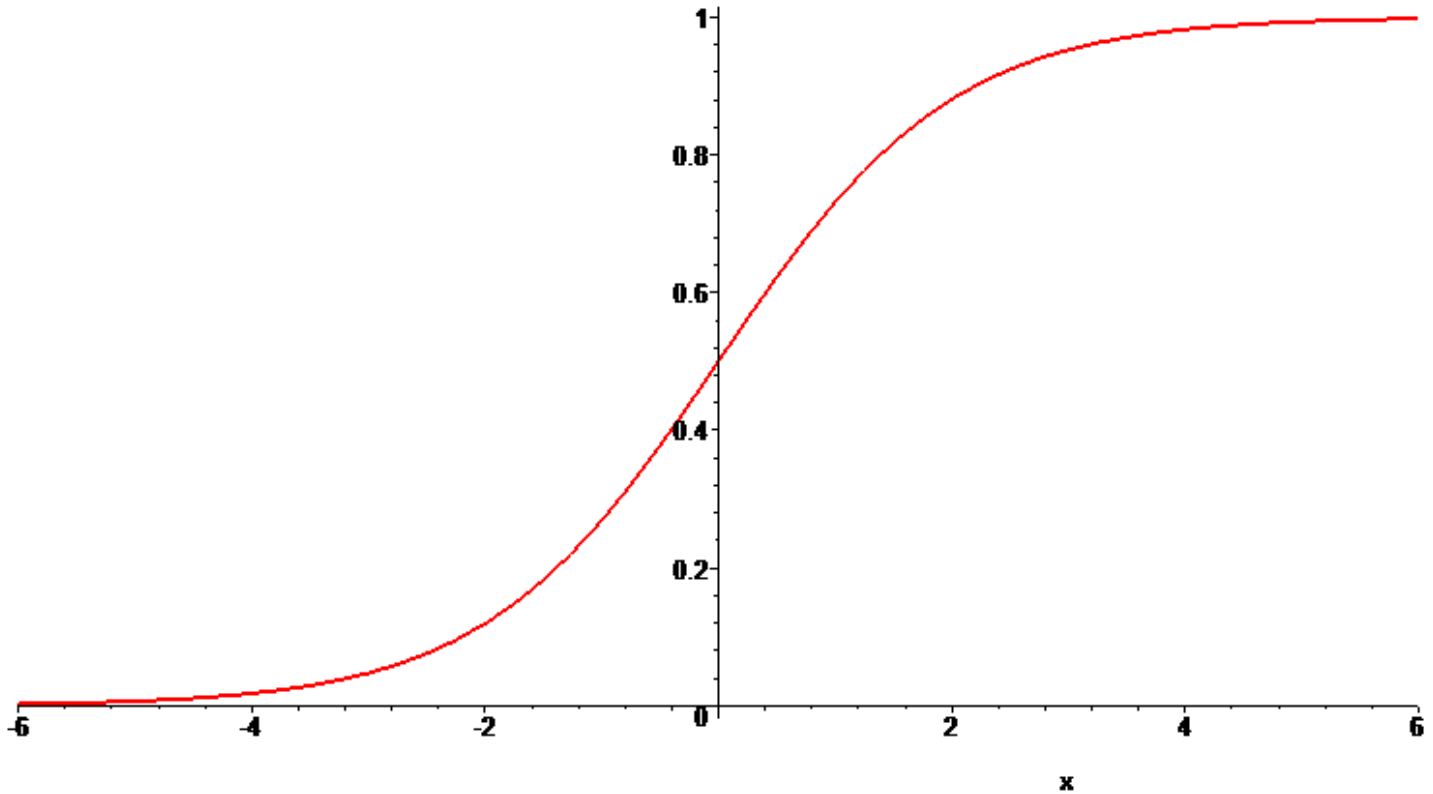
$$W(x_1, \ldots, x_n) = f(w_0 + w_1 x_1 + \cdots + w_n x_n),$$

где w_i — внутренние параметры нейрона (весовые коэффициенты) и f — передаточная или активаторная функция, в данном случае «сигмоид» $f(x) = \frac{1}{1+e^{-x}}$.



Передаточная функция: сигмоид

$$f(x) = \frac{1}{1 + e^{-x}}$$



Сигмоид, таблица

x	$f(-x)$	$f(x)$
0	0.5	0.5
1	0.26894	0.73106
2	0.11920	0.88080
3	0.04743	0.95257
4	0.01799	0.98201
5	0.00670	0.99331
6	0.00247	0.99753
7	0.00091	0.99909
8	0.00033	0.99966
9	0.00012	0.99988
10	0.00005	0.99995

Обучающая матрица

Строки обучающей матрицы X являются обучающими векторами

$$X = \begin{pmatrix} X_0 \\ X_1 \\ \dots \end{pmatrix} = \begin{pmatrix} x_{01} & \dots & x_{0,k} \\ x_{11} & \dots & x_{1,k} \\ \dots & & \end{pmatrix}$$

Для каждого обучающего вектора X_i мы знаем верный ответ Y_i — либо 0, либо 1.
Наша задача: подобрать параметры нейрона $(w_0, w_1, \dots, w_k)$ так, чтобы $Y_i \approx W(x_1, \dots, x_n)$. Более точно, чтобы сумма квадратов отклонений была бы минимальной:

$$S = \sum S_i = \sum |W(X_i) - Y_i|.$$

Минимизация функции потерь

Таким образом, задача сводится к минимизации функции (потерь)

$$S = \sum S_i = \sum |W(X_i) - Y_i|.$$

Так как

$$W(X_i) = W(x_{i,1}, \ldots, x_{i,n}) = f(w_0 + w_1 x_{i1} + \cdots + w_n x_{in}),$$

то

$$S = \sum |f(w_0 + w_1 x_{i1} + \cdots + w_n x_{in}) - Y_i|.$$

где сумма берётся по всем строкам обучающей матрицы

$$X_i = (x_{i1}, \ldots, x_{in})$$

и соответствующим значениям Y_i, то есть

$$S = S(w_0, w_1, \ldots, w_n).$$

Минимизация функции потерь

В нашем примере матрицы Y, X имеют вид:

```
A: 0.00 0.00 7.00, B:10.0010.00 7.00; 1000 points
A,  -4.8627,  -2.7670
B,  15.1039,   6.7192
A,  -6.1315,  -0.5620
B,   8.7361,   9.4905
...
```

то есть $n = 2$ и S является функцией от 3-х переменных:

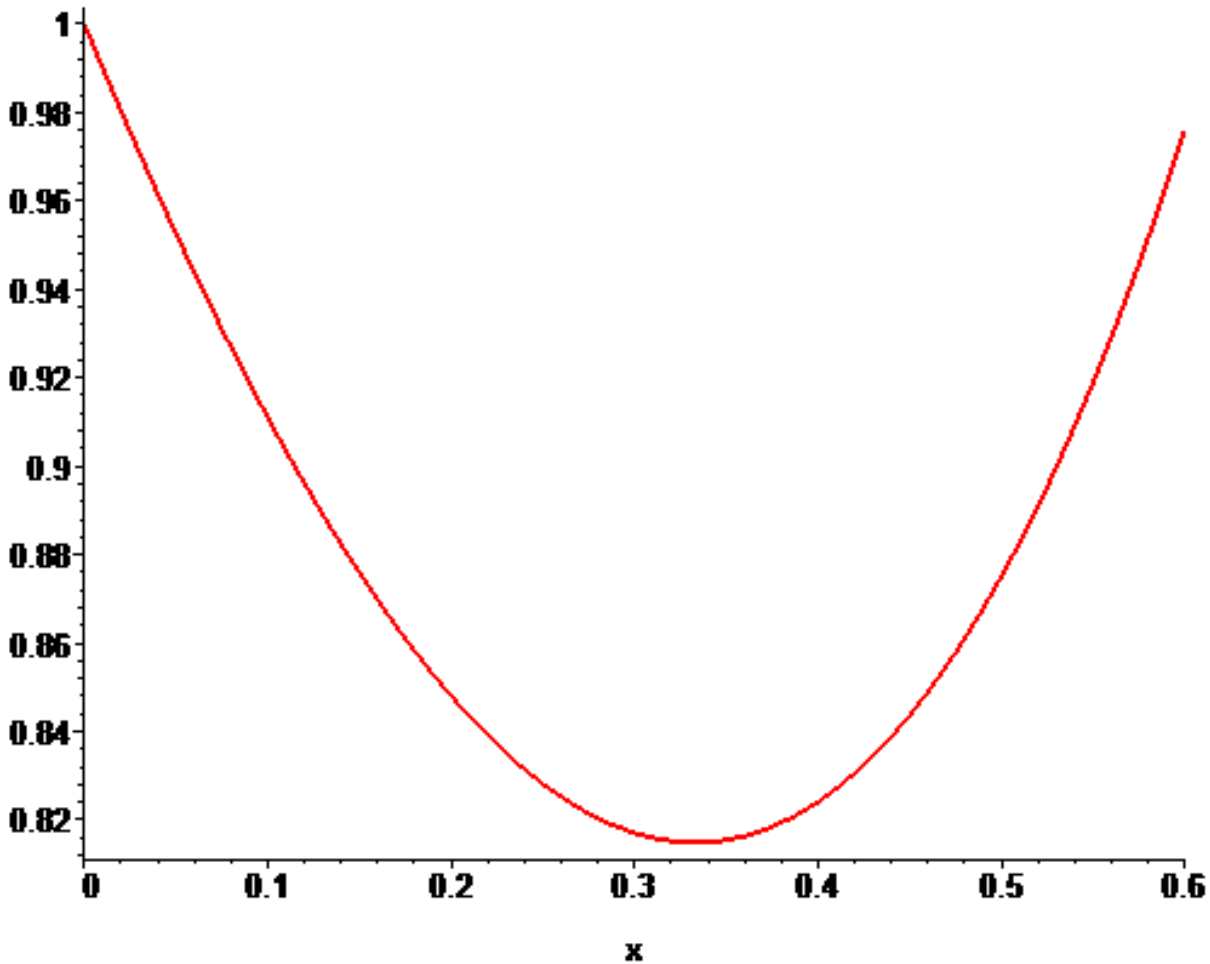
$$S = S(w_0, w_1, w_2).$$

Минимизация в размерности 1

Для примера, найти минимум функции:

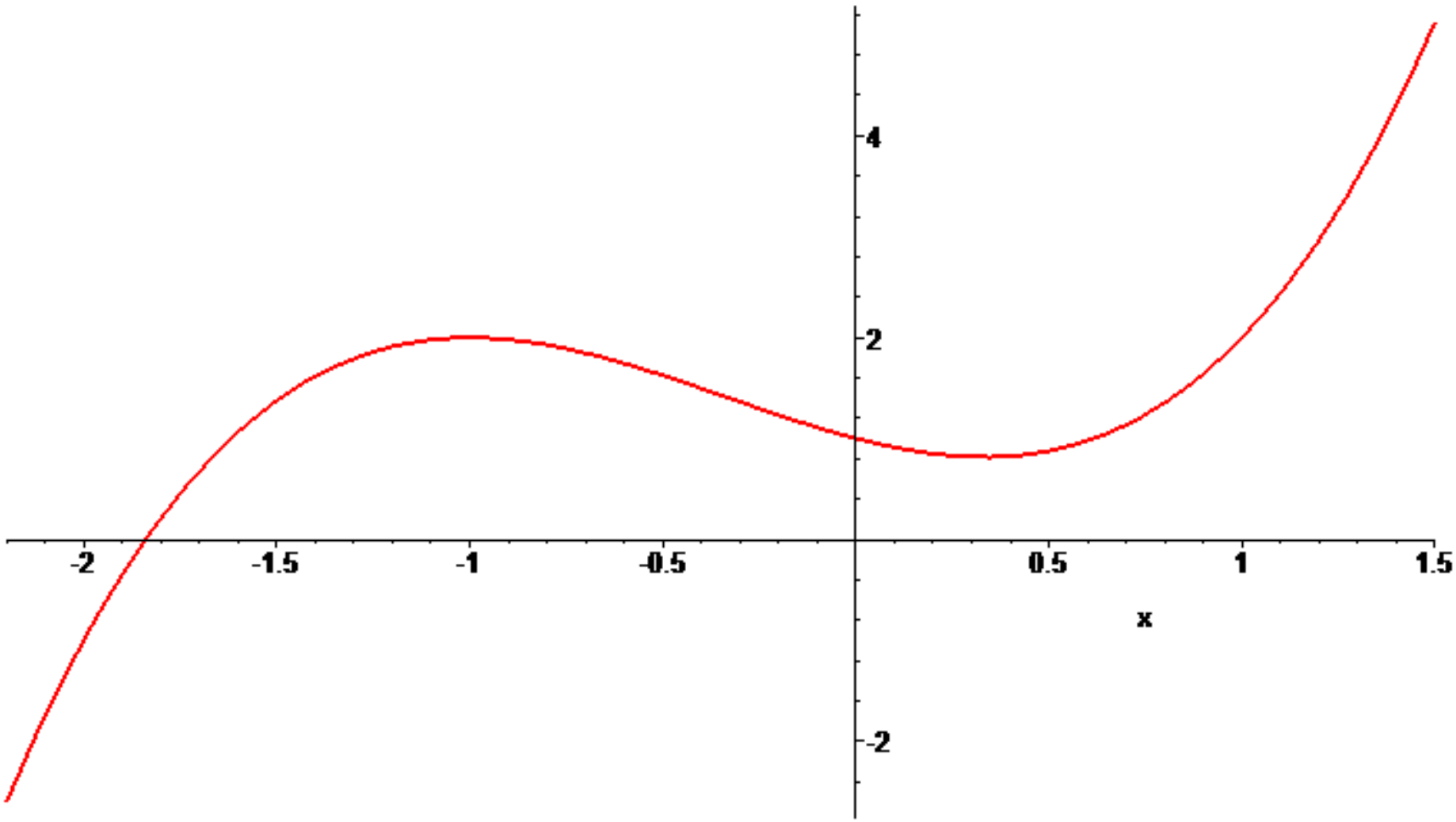
$$f(x) = 1 - x + x^2 + x^3$$

на отрезке [0,1].



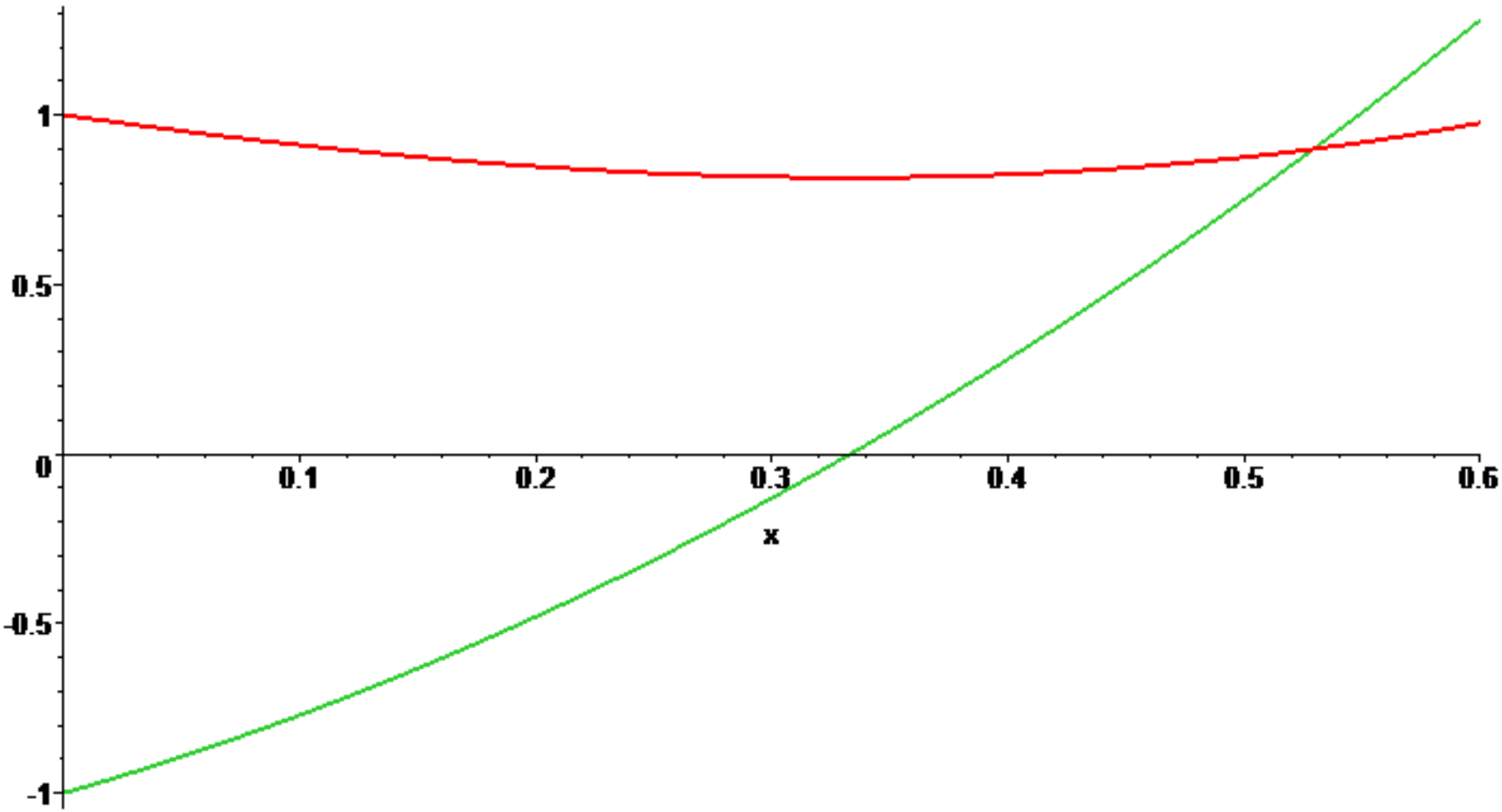
Минимизация в размерности 1

На самом деле, за пределами отрезка [0,1] функция ведёт себя так:



Минимизация в размерности 1

Покажем на одном графике и функцию $f(x)$ и её производную $f'(x)$:



Минимизация в размерности 1

```
def f(x): return 1-x+x**2+x**3
def df(x): return -1+2*x+3*x**2

def gradient_descent(f, df, x0, LR):
    '''
    Нахождение минимума функции одной переменной
    методом градиентного спуска
    :param f: минимизируемая фукнция
    :param df: её производная
    :param x0: начальное значение
    :param LR: скорость обучения
    :return:
    '''
```

Минимизация в размерности 1

```
def gradient_descent(f, df, x0, LR):
  eps = 1e-8  # минимальное улучшение за однин шаг
  S0 = f(x0)
  for iStep in range(100):
    x1 = x0 - LR*df(x0)
    if x1<0 or x1>1: break
    S1 = f(x1)
    if S1 > S0-eps: break # слишком малое улучшение
    print(f'{iStep:4d},{x1:9.7f},{S1:9.7f},{S0-S1:8.1e}')
    x0, S0 = x1, S1
  return x0, S0
```

Минимизация в размерности 1

```
x0 = 1
LR = 0.1
print( gradient_descent(f, df, x0, LR))
```

Задание. Для той же самой функции $f(x)$ и её производной попробуйте разные начальные точки $x_0 = (0, 1, 0.5, 2, \dots)$ и различную скорость обучения

$$LR = (1,\ 0.5,\ 0.3,\ 0.2,\ 0.15,\ 0.1,\ 0.01,\ 0.001,\ 0.0001,\ \dots).$$

Вычисление производной

По определению:

$$f'(x) = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}$$

Для приближённого вычисления выбираем достаточно малое ε и полагаем

$$f'(x) \approx \frac{f(x+\varepsilon) - f(x)}{\varepsilon}$$

Замечание. Не стоит брать ε уж слишком малым. В задачах машинного обучения вполне достаточно взять $\varepsilon = 10^{-6}$.

Численное вычисление производной

```
def df_num(f, x, eps=1e-6):
    ''' численное вычисление производной
    :param f: дифференцируемая функция
    :param x: точка
    :param eps: epsilon
    :return: f'(x)
    eps - параметр по умолчанию!
    '''
    return (f(x+eps)-f(x))/eps

for i in range(-20, 20):
  x = i/10
  print(f'{x:5.1f},{df(x):7.4f},{df_num(f, x)-df(x):7.2e}')
```

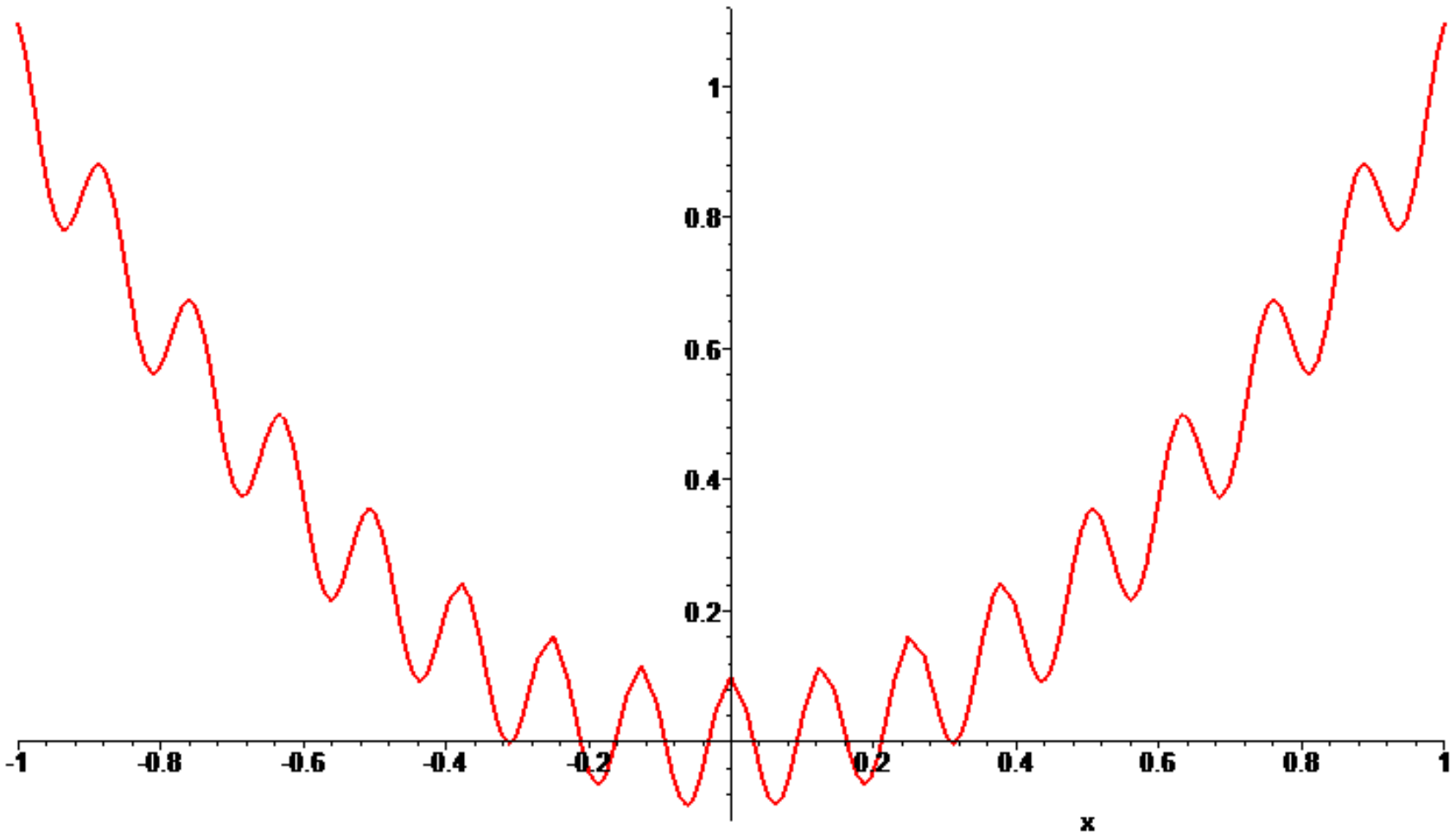
Задание

Найдите минимум функции, но с заменой точной производной на численно найденную.

```
def df1(x): return df_num(f,x)
```

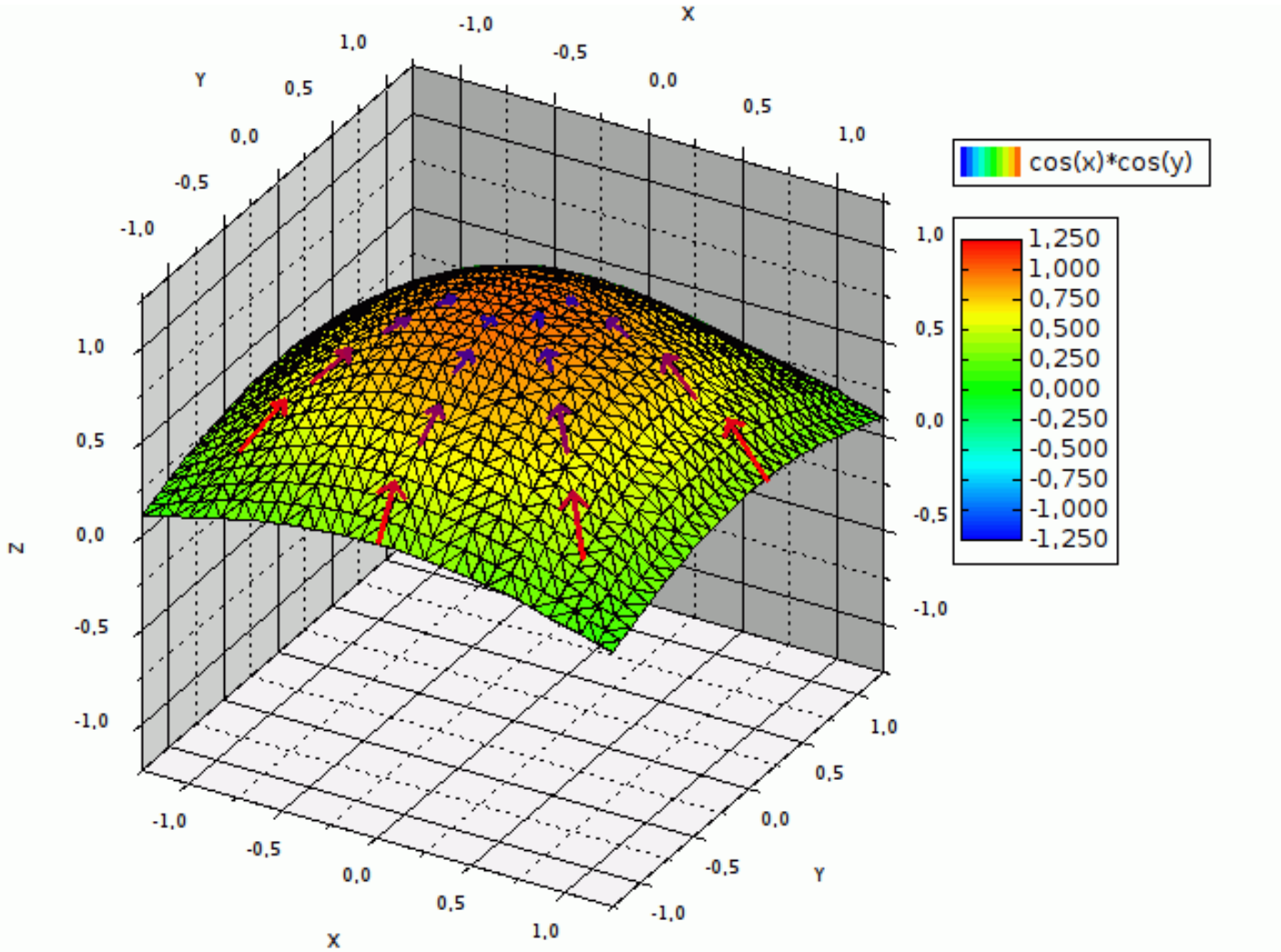
Проблема локальных минимумов

$$f(x) = x^2 + 0.1 \cos 50x$$



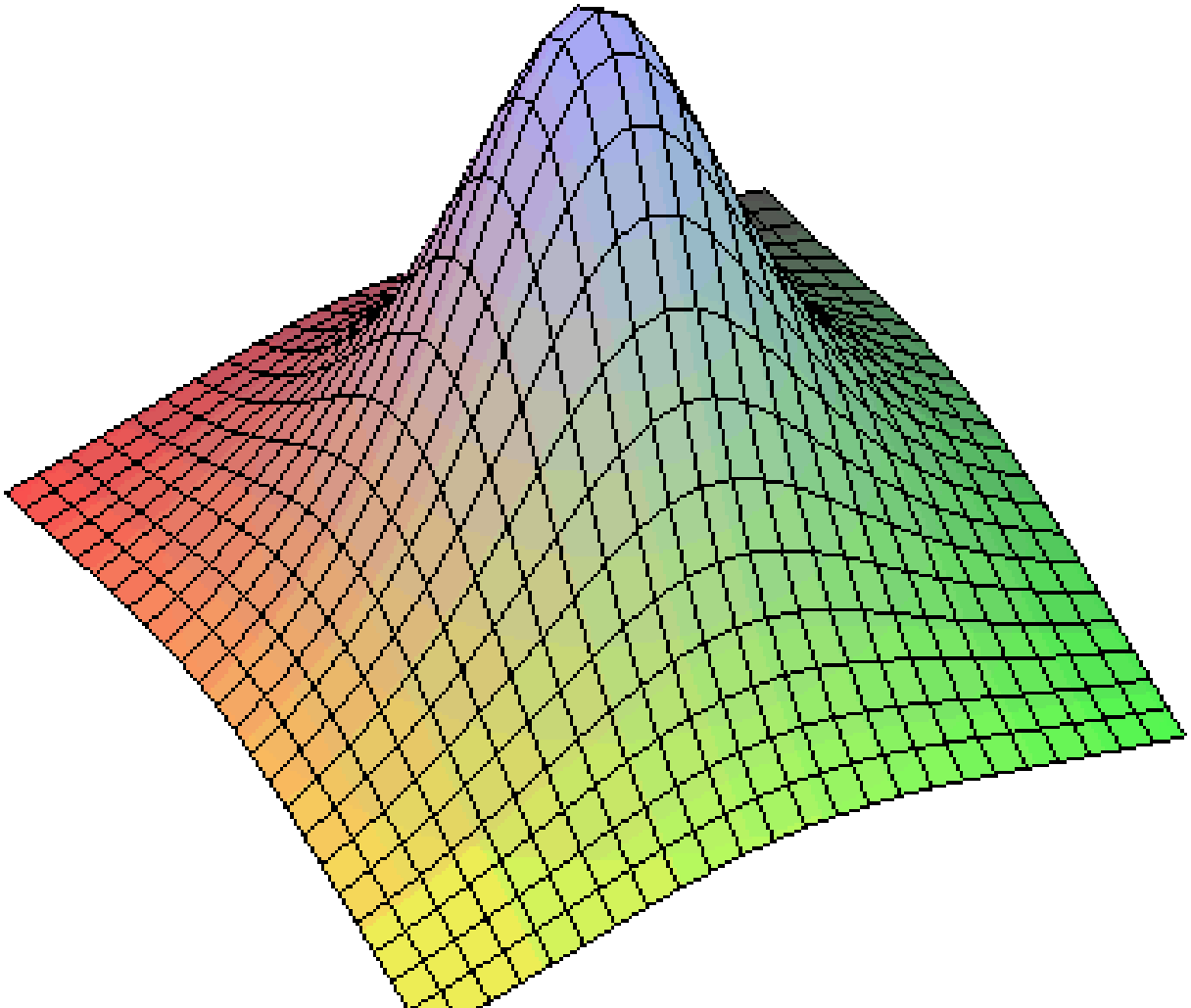
Градиент

$$f(x, y) = \cos x \cos y$$



Градиент

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

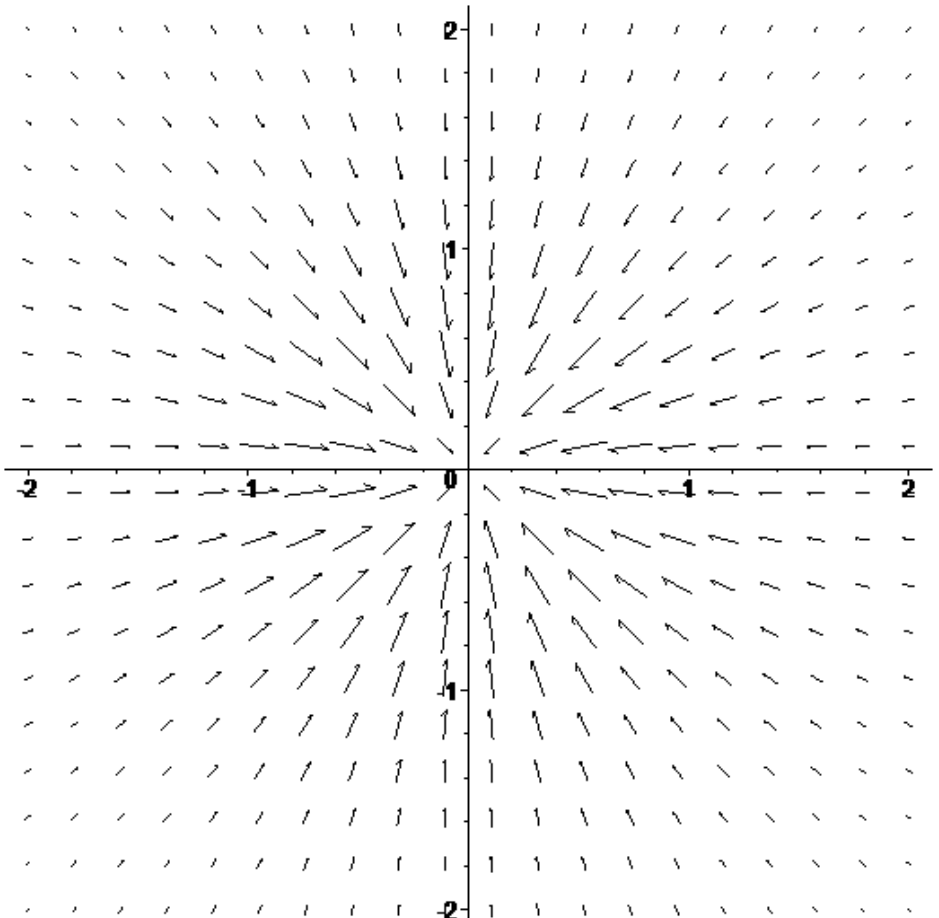


Градиент

Вектор градиента

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

показывает направление наибольшего роста функции $f(x, y)$



Две пробные функции

```
def f1(x,y): return x**2 + y**2
def df1(x,y): return [???, ???]

def f2(x,y): return 5-8*x-2*y+5*x**2-2*x*y+2*y**2
def df2(x,y): return [???, ???]

print(f1(1, 0.1))
print(df1(1, 1))
```

Минимизация

```
def gradient_descent2(f, df, x0, y0, LR):
    '''
    Нахождение минимума функции двух переменных
    методом градиентного спуска
    :param f: минимизируемая фукнция
    :param df: её производная
    :param x0, y0: начальная точка
    :param LR: скорость обучения
    :return: найденные x,y и f(x,y)
    '''
```

Минимизация

```
def gradient_descent2(f, df, x0, y0, LR):
  eps = 1e-8  # минимальное улучшение за один шаг
  S0 = f(x0,y0)
  for iStep in range(100):
    dx, dy = df(x0, y0)
    x1, y1 = x0 - LR*dx, y0 - LR*dy
    if x1<-5 or x1>5: break
    if y1<-5 or y1>5: break
    S1 = f(x1,y1)
    if S1 > S0-eps: break # слишком малое улучшение
    print(f'{iStep:4d}, pnt({x1:9.7f}, {y1:9.7f}),',
          f'S={S1:9.7f}, dS={S0-S1:8.1e}')
    x0, y0, S0 = x1, y1, S1
  return x0, y0, S0
```

Задание

Задание. Для функций $f1(x, y)$, $f2(x, y)$ попробуйте разные начальные точки и различную скорость обучения

$$LR = (1,\ 0.5,\ 0.3,\ 0.2,\ 0.15,\ 0.1,\ 0.01,\ 0.001,\ 0.0001,\ \ldots).$$

```
for LR in (1, 0.4, 0.2, 0.1, 0.01, 0.0001):
    ...
```

При этот отладочный print внутри функции лучше отключить. Измените функцию gradient_descent так, чтобы она возвращала ещё и количество шагов.

Функция Розенброка

Реализуйте функцию Розенброка и её градиент:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Она имеет минимум 0 в точке (1,1).

```
def Rozenbrok(x,y):  return ???
def dRozenbrok(x,y): return [???, ???]

print(Rozenbrok(1,1))
print(dRozenbrok(1,1))
```

Численное вычисление производной

```
def df_num2(f, x, y, eps=1e-6):
  '''
  численное вычисление производной
  :param f: дифференцируемая функция (x,y)
  :param x,y: точка
  :param eps: epsilon
  :return: [df/dx, df/dy]
  '''
  fxy = f(x,y)
  return [(f(x+eps,y)-fxy)/eps, (f(x,y+eps)-fxy)/eps]
```

Проверить для функций $f_1(x, y)$, $f_2(x, y)$.

Задание

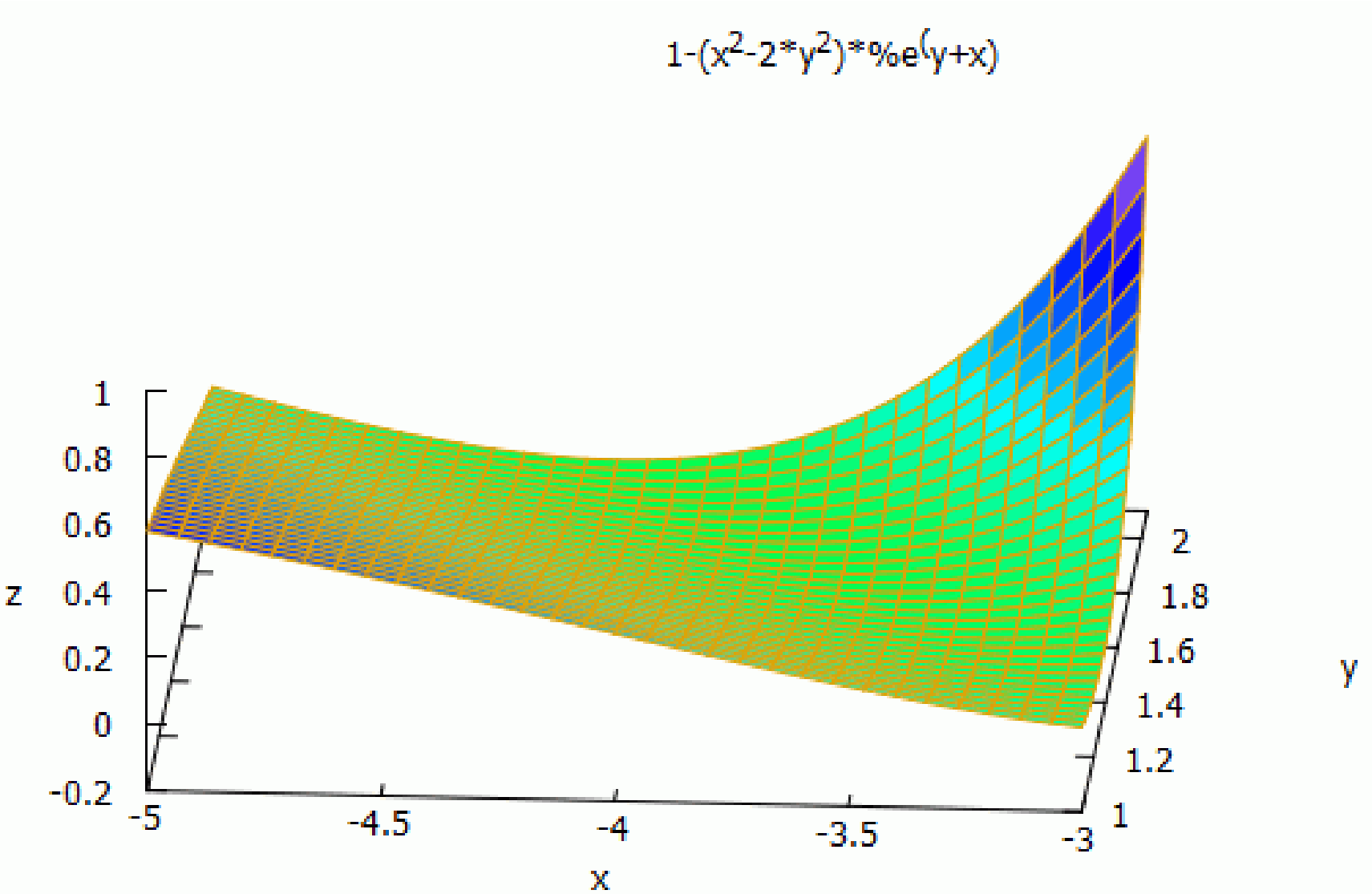
Найдите минимум функции

$$f_3(x, y) = 1 - e^{x+y}(x^2 - 2y^2)$$

На Максиме:

```
f(x,y):=1-%e^(x+y)*(x^2-2*y^2);
plot3d(f(x,y),[x,-5,-3],[y,1,2.1])$
```

Задание



Подсказка: в точке (-4,2).

Минимизация функции от n переменных

```
def gradient_descent(f, df, w0, LR):
  '''
  Нахождение минимума функции n переменных
  методом градиентного спуска
  :param f: минимизируемая фукнция
  :param df: её производная
  :param w0: начальная точка
  :param LR: скорость обучения
  :return: найденную точку w и f(w)
  '''
```

Минимизация функции от n переменных

```
def gradient_descent(f, df, w0, LR):
  eps = 1e-10  # минимальное улучшение за один шаг
  w0 = np.array(w0)
  S0 = S1 = f(w0)
  iStep=-1
  for iStep in range(100):
    w1 = w0 - LR*np.array(df(w0))
    if np.linalg.norm(w1)>10: break
    S1 = f(w1)
    if S1 > S0-eps: break # слишком малое улучшение
    print(f'{iStep:4d}, {w1}, {S1:13.10f}, {S0-S1:8.1e}')
    w0, S0 = w1, S1
  return w0, S0, iStep
```

Проверить для функций $f_3(x, y)$, $df_3(x, y)$.

Обучение одного нейрона

У нас имеется матрица из обучающих векторов X размера $N \times 2$ и вектор ответов Y той же длины.
Наша задача — найти три числа (w_0, w_1, w_2), минимизирующие функцию потерь

$$S(w_0, w_1, w_2) = \sum S_i = \sum |f(w_0 + w_1 x_{i0} + w_2 x_{i1}) - y_i|$$

где (x_{i0}, x_{i1}) — i-я строка матрицы X и $f(t)$ — передаточная функция «сигмоид»:

$$f(t) = \frac{1}{1 + e^{-t}}, \quad f'(t) = \frac{e^{-t}}{(1 + e^{-t})^2}.$$

Частные производные

$$S_i = |f(w_0 + w_1 x_{i0} + w_2 x_{i1}) - y_i|$$

или

$$S_i = \begin{cases} f(w_0 + w_1 x_{i0} + w_2 x_{i1}) & \text{если } y_i = 0 \\ 1 - f(w_0 + w_1 x_{i0} + w_2 x_{i1}) & \text{если } y_i = 1 \end{cases}$$

$$\frac{\partial S_i}{\partial w_0} = \pm f'(..)$$

$$\frac{\partial S_i}{\partial w_1} = \pm f'(..) \cdot x_{i0}$$

$$\frac{\partial S_i}{\partial w_2} = \pm f'(..) \cdot x_{i1}$$

где знак «+» выбирается, если $y_i == 0$ и «-» иначе.

Обучение одного нейрона

```
def Sigmoid(x): return 1/(1+np.exp(-x))

def Si(w, Xi, Yi):
    fw = Sigmoid(w[0]+np.dot(w[1:],Xi))
    return fw if Yi==0 else 1-fw

def dSi(w, Xi, Yi):
    t = np.exp(-w[0]-np.dot(w[1:],Xi))
    fs = t/(1+t)**2
    if Yi==1: fs = -fs
    # с добавленной 1 в начале массива
    return fs*np.insert(Xi, 0, 1)
```

Обучение одного нейрона

```
def S(w):   # функция потерь
    global X, Y
    s = 0
    for i, Xi in enumerate(X):
        s += Si(w, Xi, Y[i])
    return s/len(X)

def dS(w):   # градиент функции потерь
    global X, Y
    ds = np.zeros(X.shape[1]+1)
    for i, Xi in enumerate(X):
        ds += np.array(dSi(w, Xi, Y[i]))
    return ds/len(X)
```

Обучение одного нейрона

```
Y,X = load_data_01('neuron_AB.csv')
LR = 0.1
w0 = np.array((1,2,3))
res = gradient_descent(S, dS, w0, LR)
print(f'LR={LR:8.6f} steps={res[2]:6d} ',
      f'F({res[0]}) = {res[1]:8.2e}')
```

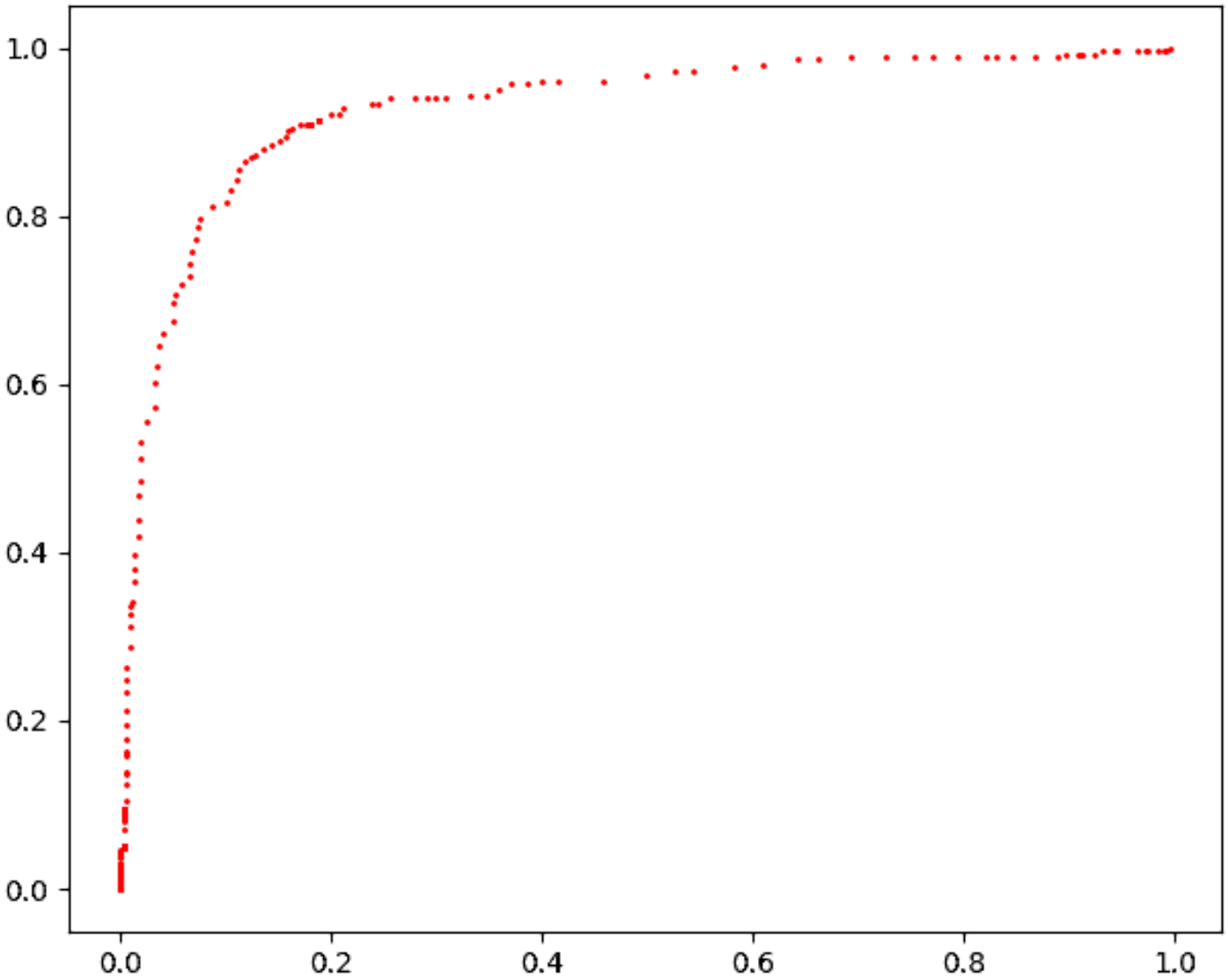
На примере UCI_Australian.csv

```
def UCI_Australian(): # работаем на файле UCI_Australian
    global Y,X
    csv = np.loadtxt("UCI_Australian.csv", skiprows=1,
                     delimiter=',')
    print(csv.shape)
    Y = csv[:, 0]; X = csv[:, 1:]
    mx = X.shape[1]
    # нормализуем столбцы матрицы X
    X -= X.mean(axis=0) # Из каждого столбца матрицы X
                        # вычесть его среднее значение
    X /= X.std(axis=0)  # Каждый столбец матрицы X поделить
                        # на его ср.кв.отклонение
    LR = 0.5
    w0 = np.linspace(0,1, mx+1)
    res = gradient_descent(S, dS, w0, LR)
    print(res)
```

На примере UCI_Australian.csv, ROC кривая

```
def UCI_Australian(): # работаем на файле UCI_Australian
    ...
    print(res)
    w = res[0]       # рассчитанный вектор w
    cx, cy = [], []
    for ib in range(-60, 150):
        TP, FP, FN, TN = TP_FP_FN_TN(X, Y, w, ib/6)
        cx.append(FP / (FP + TN))
        cy.append(TP / (TP + FN))
    print(f'AUC = {AUC(cx, cy): 7.4f}')
    plt.scatter(cx, cy, s=1, color='red')
    plt.show()
```

UCI_Australian.csv, ROC кривая



$AUC = 0.928$. Напомню, что раньше было 0.933.