In [38]:
```
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import display
from collections.abc import Callable
import pandas
```

Описание исходной функции $\sigma(\nu(X,W))$

In [39]:
```
def sigmoid(x: np.ndarray):
    return 1/(1+np.exp(-x))
def func(X: np.ndarray,W: np.ndarray):
    return sigmoid(np.dot(X,W))
```

Теперь найдем производную данной функции согласно ранее описанному правилу

In [40]:
```
def deriv(func: Callable[[np.ndarray],np.ndarray],X: np.ndarray,delta: float = np.power(
    return (func(X+delta)-func(X-delta))/(2*delta)
def derivFunc(X: np.ndarray,W: np.ndarray):
    #ищем производную соглавно ранее написанной теории
    dSdN = deriv(sigmoid,np.dot(X,W))
    return np.dot(dSdN,W.T)
```

Проверим результат согласно определению дифференцируемости

In [41]:
```
Xstart = (np.random.random([3,3])-0.5)*5
W = (np.random.random([3,2])-0.5)*5

pointStart = (np.random.random()-0.5)*4
delta = 0.01
data = {"jk": [],
        "sumDeltaF": [],
        "sumDifferential": []}

for j in range(Xstart.shape[0]):
    for k in range(Xstart.shape[1]):
        # фиксируем все x, кроме x_jk
        XpointStart = Xstart.copy()
        XpointFinish = Xstart.copy()
        XpointStart[j][k] = pointStart
        XpointFinish[j][k] = pointStart+delta
        deltaF = -func(XpointStart,W) + func(XpointFinish,W)
        deltaX = np.zeros_like(Xstart)
        deltaX[j][k] = delta
        gradient = derivFunc(XpointStart,W)
        differential = deltaX*gradient


        data['jk'].append(str(j+1)+str(k+1))
        data['sumDeltaF'].append(np.sum(deltaF))
        data['sumDifferential'].append(np.sum(differential))

dataFrame = pandas.DataFrame(data)

display(dataFrame)
```

	jk	sumDeltaF	sumDifferential
0	11	0.000394	0.000406
1	12	-0.003132	-0.003148
2	13	0.000200	0.000199
3	21	0.000830	0.000820
4	22	-0.008398	-0.008364
5	23	-0.003385	-0.003359
6	31	-0.000623	-0.000617
7	32	-0.002065	-0.002074
8	33	0.000022	0.000023