

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра математического моделирования и искусственного интеллекта

## ОТЧЕТ ПО КОНТРОЛЬНОЙ РАБОТЕ № 4

Дисциплина: Методы машинного обучения

Студент: Петров Артем Евгеньевич

Группа: НКНбд-01-21

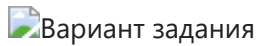
Москва 2024

---

1. Постройте тензор ранга 1 (вектор) со значениями заданной в индивидуальном задании функции одной переменной на заданном в индивидуальном задании отрезке и определите максимальное и минимальное значения функции.
2. Постройте график функции с прямыми, соответствующими максимальному и минимальному значениям, подписывая оси и рисунок и создавая легенду.
3. Найдите значения производной от функции порядка, указанного в индивидуальном задании, и построьте график полученной функции, подписывая оси и рисунок.
4. Постройте тензор ранга 2 (матрицу) со значениями заданной в индивидуальном задании функции двух переменных на заданном в индивидуальном задании прямоугольнике и определите максимальное и минимальное значения функции.
5. Постройте 3d график поверхности функции двух переменных, подписывая оси и рисунок.
6. Найдите значения смешанной производной от функции порядка, указанного в индивидуальном задании, и построьте 3d график поверхности полученной функции, подписывая оси и рисунок.
7. Решите задачу парной линейной регрессии при помощи модели TensorFlow, рассматривая тензор ранга 1 из пункта 1 задания как значения зависимой переменной (отклика), а точки отрезка из индивидуального задания как значения независимой переменной (предиктора). Предварительно масштабируйте независимую и зависимую переменные на интервал  $[0, 1]$ . Оцените качество полученной модели по показателю качества регрессии, указанному в

индивидуальном задании. Количество эпох, скорость обучения и начальные значения весов выберите самостоятельно, обеспечивая сходимость обучения.

8. Постройте кривую обучения для показателя качества регрессии, указанного в индивидуальном задании, с зависимостью от количества эпох. Показатель качества регрессии реализуйте как функцию с использованием функций модуля `tf.math`.
9. Изобразите на графике точки набора данных (независимой и зависимой переменных) и линию построенной парной регрессии, подписывая оси и рисунок и создавая легенду.



## 1. Постройте тензор ранга 1 (вектор) со значениями заданной в индивидуальном задании функции одной переменной на заданном в индивидуальном задании отрезке и определите максимальное и минимальное значения функции.

$$f(x) = (3 \cdot x - x^3) \cdot \sin(x), x \in [0, 2]$$

```
In [178... import tensorflow as tf
import math
import numpy as np
```

```
In [179... X = np.arange(0, 2, 0.01)
```

```
In [180... tensor_xy = tf.constant([X, [(3*x - x ** 3) * np.sin(x) for x in X]], dtype = tf.float64)
print(tf.rank(tensor_xy))
```

```
tf.Tensor(2, shape=(), dtype=int32)
```

```
In [181... print("MAX: ", tensor_xy[0][tf.argmax(tensor_xy[1])], tf.reduce_max(tensor_xy[1]))
print("MIN: ", tensor_xy[0][tf.argmin(tensor_xy[1])], tf.reduce_min(tensor_xy[1]))
```

```
MAX: tf.Tensor(0.99, shape=(), dtype=float32) tf.Tensor(4.9600000000000035, shape=(), dtype=float64)
```

```
MIN: tf.Tensor(0.0, shape=(), dtype=float32) tf.Tensor(1.0, shape=(), dtype=float64)
```

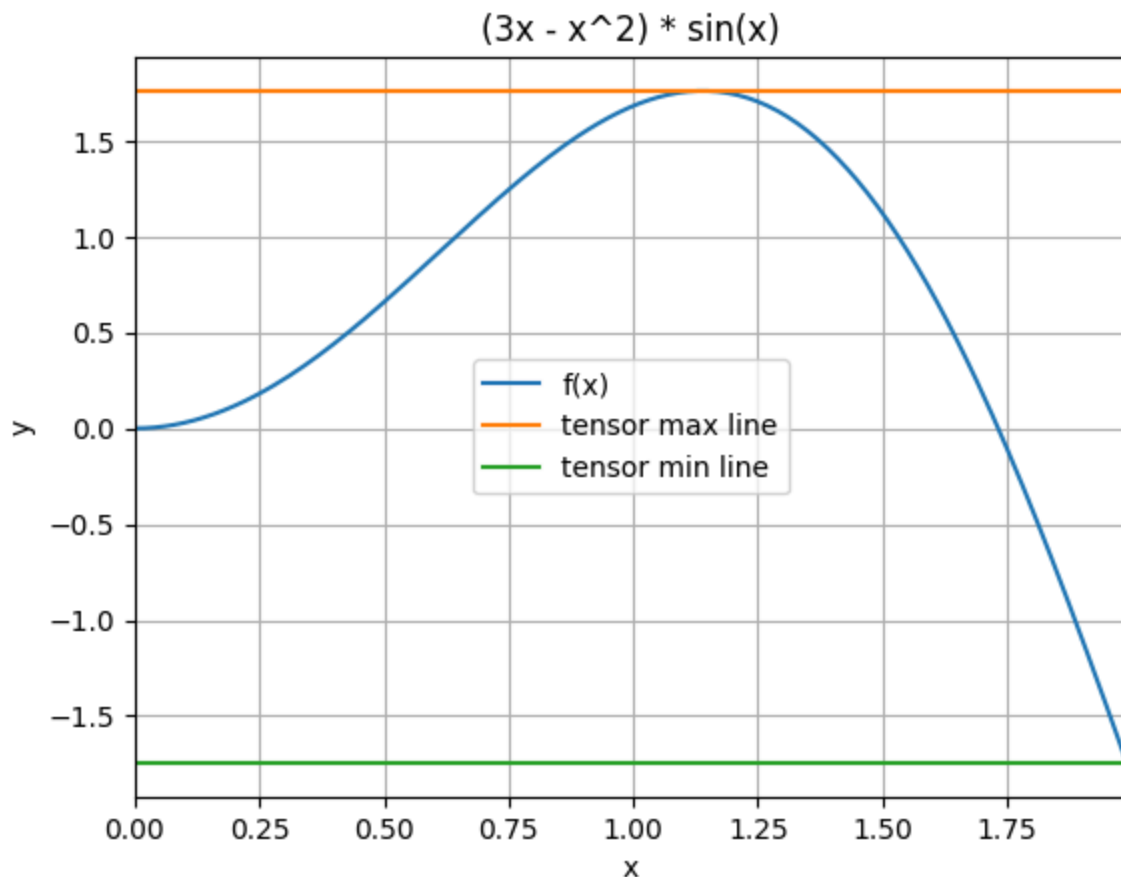
## 2. Постройте график функции с прямыми, соответствующими максимальному и минимальному значениям, подписывая оси и рисунок и создавая легенду.

```
In [182... import matplotlib.pyplot as plt
```

```
plt.rcParams.update({
    'text.usetex' : False
})
```

```
In [185... plt.plot(tensor_xy[0], tensor_xy[1], label = 'f(x)')
plt.grid(True)
plt.plot([0, 2], [tf.reduce_max(tensor_xy[1]), tf.reduce_max(tensor_xy[1])], label
plt.plot([0, tensor_xy[0][tf.argmax(tensor_xy[1])], [tf.reduce_min(tensor_xy[1]),
plt.xlim([0, max(X)])
plt.title('(3x - x^2) * sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

Out[185... <matplotlib.legend.Legend at 0x1d1e1213800>



**3. Найдите значения производной от функции порядка, указанного в индивидуальном задании, и постройте график полученной функции, подписывая оси и рисунок.**

Порядок ф-ции одной производной -- 4

```
In [186... X_tensor = tf.Variable(tensor_xy[0])
# Y_tensor = tf.Variable(tensor)
```

```

with tf.GradientTape() as der1:
    with tf.GradientTape() as der2:
        with tf.GradientTape() as der3:
            with tf.GradientTape() as der4:
                Y_tensor = (3*X_tensor - X_tensor**2) * np.sin(X_tensor)
                d4ydx4 = der4.gradient(Y_tensor, X_tensor)
                print("Y=", d4ydx4[0:5], "\n", "X=", X_tensor[0:5])

```

```

Y= tf.Tensor([0.          0.0297995  0.05919605 0.08818676 0.11676885], shape=(5,), dtype=float32)
X= tf.Tensor([0.   0.01 0.02 0.03 0.04], shape=(5,), dtype=float32)

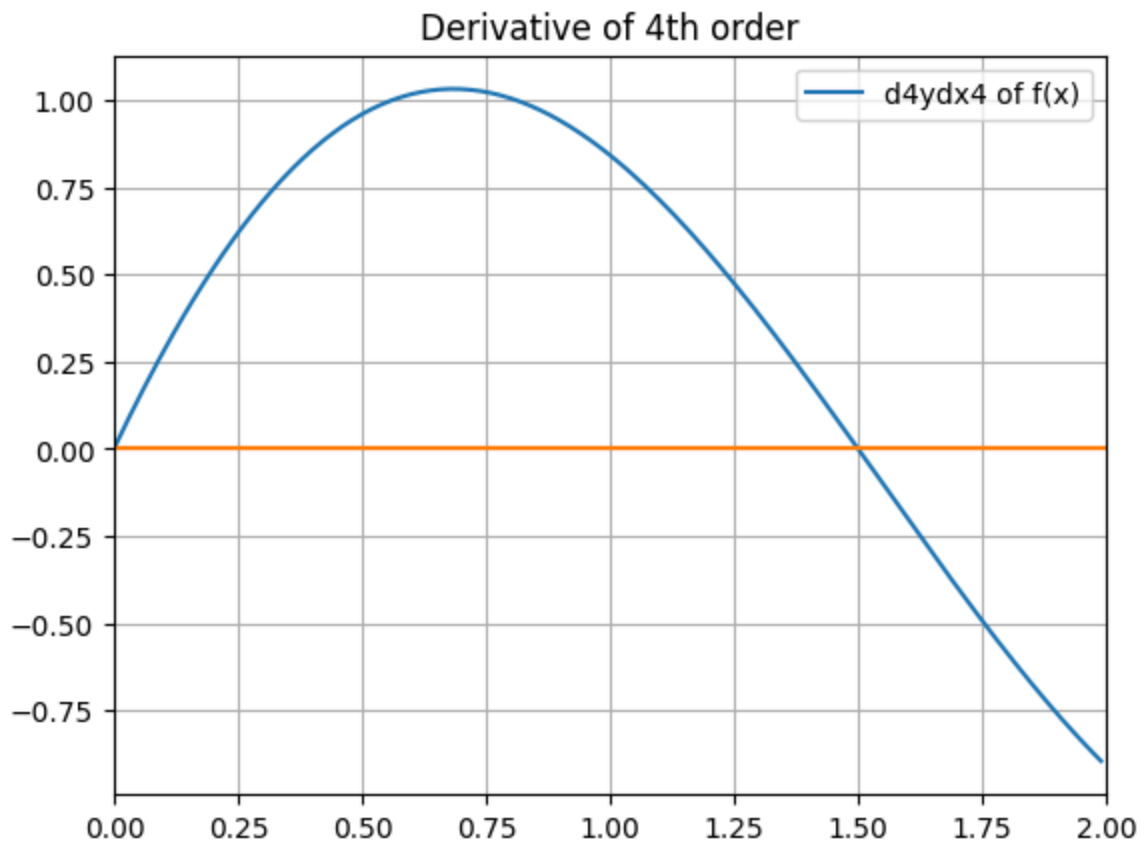
```

```

In [187... plt.plot(X_tensor, d4ydx4, label = "d4ydx4 of f(x)")
plt.grid(True)
plt.title('Derivative of 4th order')
plt.legend()
plt.plot([0, 3], [0, 0])
plt.xlim(0, 2)

```

```
Out[187... (0.0, 2.0)
```



```

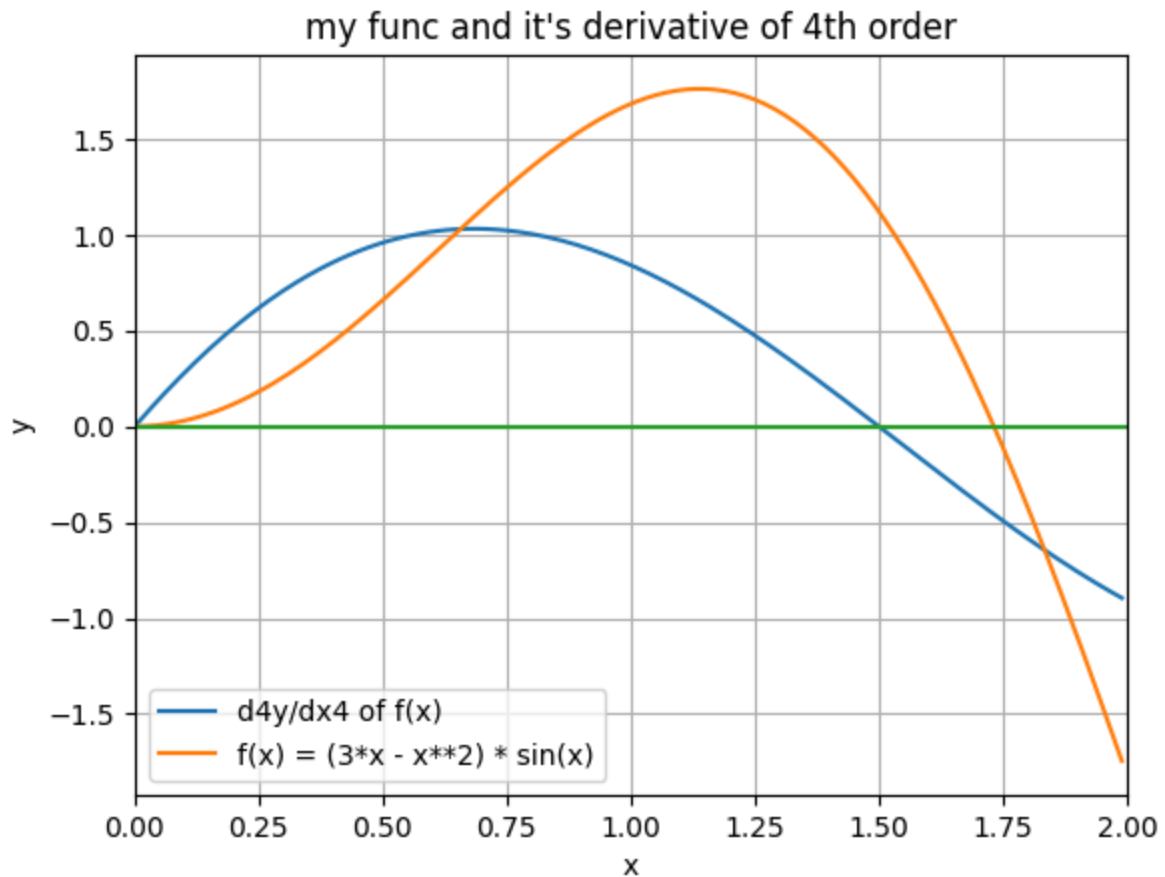
In [188... plt.plot(X_tensor, d4ydx4, label = "d4y/dx4 of f(x)")

plt.plot(tensor_xy[0], tensor_xy[1], label = 'f(x) = (3*x - x**2) * sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.title('my func and it\'s derivative of 4th order')
plt.legend()

```

```
plt.plot([0, 3], [0, 0])
plt.xlim(0, 2)
```

Out[188... (0.0, 2.0)



In [189...   
`del der1`  
`del der2`  
`del der3`  
`del der4`

**4. Постройте тензор ранга 2 (матрицу) со значениями заданной в индивидуальном задании функции двух переменных на заданном в индивидуальном задании прямоугольнике и определите максимальное и минимальное значения функции.**

$f(x, y) = x \ln(x+y)$ ,  $x$  in  $[1, 5]$  and  $y$  in  $[1, 5]$

```
In [190... X = [x for x in np.arange(1, 5, 0.04)]  
Y = [y for y in np.arange(1, 5, 0.04)]  
print(len(X), len(Y))
```

100 100

```
In [191... import itertools
```

```
In [192... tensor = tf.constant([X, Y, [x * np.log(x+y) for x, y in zip(X, Y)]], dtype = tf.fl  
print(tensor)
```

```
tf.Tensor(
[[ 1.      1.04      1.08      1.12      1.16      1.2
  1.24      1.28      1.32      1.36      1.4       1.44
  1.48      1.52      1.56      1.6       1.64      1.68
  1.72      1.76      1.8       1.84      1.88      1.92
  1.96      2.       2.04      2.08      2.12      2.16
  2.2       2.24      2.28      2.32      2.36      2.4
  2.44      2.48      2.52      2.56      2.6       2.64
  2.68      2.72      2.76      2.8       2.84      2.88
  2.92      2.96      3.       3.04      3.08      3.12
  3.16      3.2       3.24      3.28      3.32      3.36
  3.4       3.44      3.48      3.52      3.56      3.6
  3.64      3.68      3.72      3.76      3.8       3.84
  3.88      3.92      3.96      4.       4.04      4.08
  4.12      4.16      4.2       4.24      4.28      4.32
  4.36      4.4       4.44      4.48      4.52      4.56
  4.6       4.64      4.68      4.72      4.76      4.8
  4.84      4.88      4.92      4.96      ]
[ 1.      1.04      1.08      1.12      1.16      1.2
  1.24      1.28      1.32      1.36      1.4       1.44
  1.48      1.52      1.56      1.6       1.64      1.68
  1.72      1.76      1.8       1.84      1.88      1.92
  1.96      2.       2.04      2.08      2.12      2.16
  2.2       2.24      2.28      2.32      2.36      2.4
  2.44      2.48      2.52      2.56      2.6       2.64
  2.68      2.72      2.76      2.8       2.84      2.88
  2.92      2.96      3.       3.04      3.08      3.12
  3.16      3.2       3.24      3.28      3.32      3.36
  3.4       3.44      3.48      3.52      3.56      3.6
  3.64      3.68      3.72      3.76      3.8       3.84
  3.88      3.92      3.96      4.       4.04      4.08
  4.12      4.16      4.2       4.24      4.28      4.32
  4.36      4.4       4.44      4.48      4.52      4.56
  4.6       4.64      4.68      4.72      4.76      4.8
  4.84      4.88      4.92      4.96      ]
[ 0.69314718 0.76166261 0.83171688 0.90325297 0.97621794 1.05056248
  1.12624061 1.20320929 1.28142817 1.36085936 1.44146718 1.52321802
  1.60608012 1.69002342 1.77501948 1.8610413  1.94806321 2.03606084
  2.12501093 2.21489134 2.30568092 2.39735946 2.48990764 2.58330694
  2.67753964 2.77258872 2.86843786 2.96507135 3.06247413 3.16063167
  3.25952999 3.35915562 3.45949558 3.56053733 3.66226877 3.7646782
  3.86775434 3.97148624 4.07586333 4.18087536 4.28651243 4.3927649
  4.49962345 4.60707905 4.71512289 4.82374647 4.9329415  5.04269993
  5.15301393 5.26387589 5.37527841 5.48721428 5.59967647 5.71265817
  5.8261527  5.94015357 6.05465445 6.16964918 6.28513172 6.4010962
  6.51753688 6.63444816 6.75182457 6.86966076 6.9879515  7.10669169
  7.22587634 7.34550055 7.46555956 7.58604868 7.70696334 7.82829906
  7.95005146 8.07221623 8.19478918 8.31776617 8.44114317 8.56491621
  8.68908142 8.81363498 8.93857316 9.06389231 9.18958881 9.31565916
  9.44209988 9.56890757 9.69607891 9.82361062 9.95149947 10.07974231
  10.20833603 10.33727758 10.46656396 10.59619223 10.72615948 10.85646287
  10.98709996 11.11806691 11.2493621 11.38098249]], shape=(3, 100), dtype=float64)
```

```
In [193... print("rank = ", tf.rank(tensor))
```

```
rank = tf.Tensor(2, shape=(), dtype=int32)
```

```
In [194... print("max(f(x, y)) =", tf.reduce_max(tensor[2]).numpy(), "\n", \
        "x =", tensor[0][tf.argmax(tensor[2]).numpy()], "\n", \
        "y =", tensor[1][tf.argmax(tensor[2]).numpy()])
```

```
max(f(x, y)) = 11.380982489632048
x = 4.96000000000000035
y = 4.96000000000000035
```

## 5. Постройте 3d график поверхности функции двух переменных, подписывая оси и рисунок.

```
In [195... from matplotlib import cm
```

```
In [196... x = tensor[0]
y = tensor[1]
z = tensor[2]
```

```
In [197... fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

def f(x, y):
    return x * np.log(x+y)

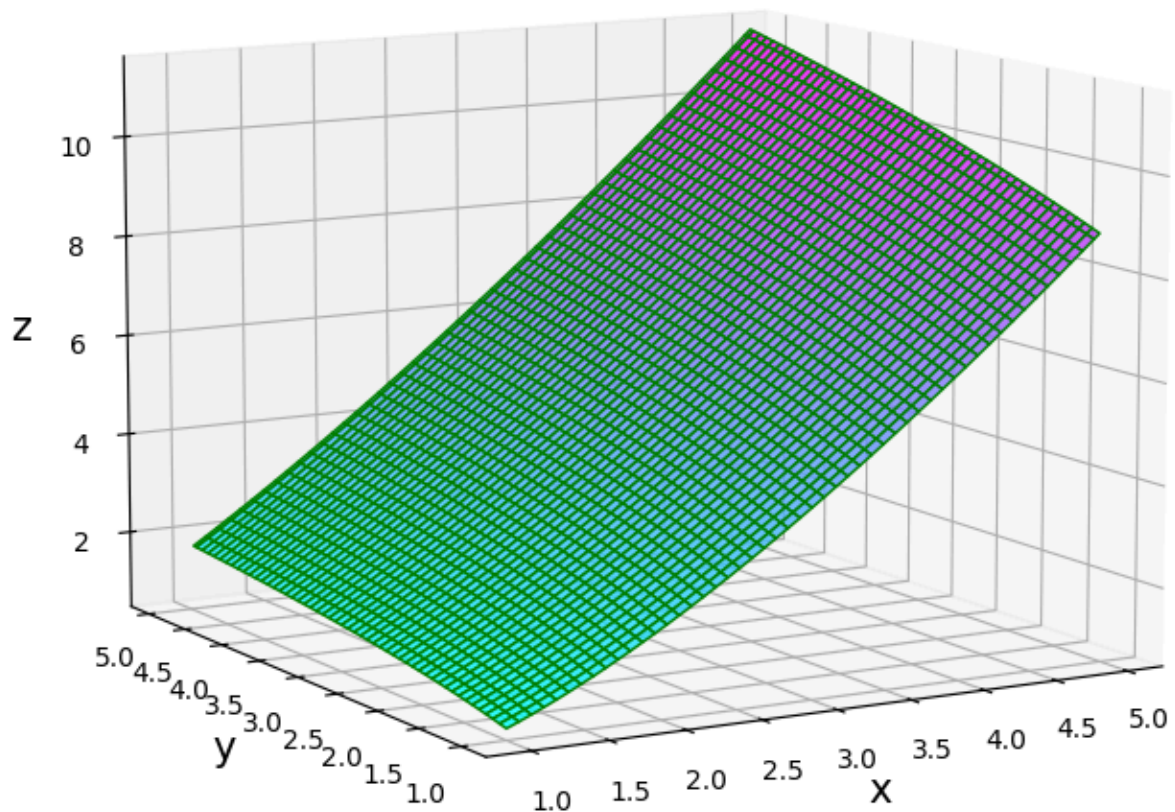
X_mesh, Y_mesh = np.meshgrid(x, y)
ax.plot_surface(X_mesh, Y_mesh, f(X_mesh, Y_mesh), color = 'green', cmap = 'cool',

ax.view_init(azim = -120, elev = 10)
ax.set_title("x*ln(x+y)")
ax.set_xlabel('x', fontsize = 15)
ax.set_ylabel('y', fontsize = 15)
ax.set_zlabel('z', fontsize = 15)
```

```
Out[197... Text(0.5, 0, 'z')
```



$$x \cdot \ln(x+y)$$



6. Найдите значения смешанной производной от функции порядка, указанного в индивидуальном задании, и постройте 3d график поверхности полученной функции, подписывая оси и рисунок.

$d^3/dx^2dy$

In [198...

```
# x = tf.Variable(tensor[0])  
# y = tf.Variable(tensor[1])  
# z = tf.Variable(tensor[2])
```

```
# x, y, z

x = tf.Variable(tensor[0])
y = tf.Variable(tensor[1])
# z = tf.Variable(tensor[2])
x, y
```

```
Out[198...] (<tf.Variable 'Variable:0' shape=(100,) dtype=float64, numpy=
array([1. , 1.04, 1.08, 1.12, 1.16, 1.2 , 1.24, 1.28, 1.32, 1.36, 1.4 ,
      1.44, 1.48, 1.52, 1.56, 1.6 , 1.64, 1.68, 1.72, 1.76, 1.8 , 1.84,
      1.88, 1.92, 1.96, 2. , 2.04, 2.08, 2.12, 2.16, 2.2 , 2.24, 2.28,
      2.32, 2.36, 2.4 , 2.44, 2.48, 2.52, 2.56, 2.6 , 2.64, 2.68, 2.72,
      2.76, 2.8 , 2.84, 2.88, 2.92, 2.96, 3. , 3.04, 3.08, 3.12, 3.16,
      3.2 , 3.24, 3.28, 3.32, 3.36, 3.4 , 3.44, 3.48, 3.52, 3.56, 3.6 ,
      3.64, 3.68, 3.72, 3.76, 3.8 , 3.84, 3.88, 3.92, 3.96, 4. , 4.04,
      4.08, 4.12, 4.16, 4.2 , 4.24, 4.28, 4.32, 4.36, 4.4 , 4.44, 4.48,
      4.52, 4.56, 4.6 , 4.64, 4.68, 4.72, 4.76, 4.8 , 4.84, 4.88, 4.92,
      4.96])>,
<tf.Variable 'Variable:0' shape=(100,) dtype=float64, numpy=
array([1. , 1.04, 1.08, 1.12, 1.16, 1.2 , 1.24, 1.28, 1.32, 1.36, 1.4 ,
      1.44, 1.48, 1.52, 1.56, 1.6 , 1.64, 1.68, 1.72, 1.76, 1.8 , 1.84,
      1.88, 1.92, 1.96, 2. , 2.04, 2.08, 2.12, 2.16, 2.2 , 2.24, 2.28,
      2.32, 2.36, 2.4 , 2.44, 2.48, 2.52, 2.56, 2.6 , 2.64, 2.68, 2.72,
      2.76, 2.8 , 2.84, 2.88, 2.92, 2.96, 3. , 3.04, 3.08, 3.12, 3.16,
      3.2 , 3.24, 3.28, 3.32, 3.36, 3.4 , 3.44, 3.48, 3.52, 3.56, 3.6 ,
      3.64, 3.68, 3.72, 3.76, 3.8 , 3.84, 3.88, 3.92, 3.96, 4. , 4.04,
      4.08, 4.12, 4.16, 4.2 , 4.24, 4.28, 4.32, 4.36, 4.4 , 4.44, 4.48,
      4.52, 4.56, 4.6 , 4.64, 4.68, 4.72, 4.76, 4.8 , 4.84, 4.88, 4.92,
      4.96])>)
```

```
In [199...] def f(x, y):
    return x * tf.math.log(x + y)

with tf.GradientTape(persistent = True) as tape:
    # tape.watch(x)
    # tape.watch(y)

    z = f(x, y)
    # tape.watch(z)

    dx =tape.gradient(z, x)
    print(dx)
    dxx = tape.gradient(dx, x)
    print(dxx)
    dxxdy = tape.gradient(dxx, y)
    print(dxxdy)

# del tape
```

```
tf.Tensor(  
[1.19314718 1.23236789 1.27010822 1.30647587 1.34156719 1.37546874  
1.40825856 1.44000726 1.47077892 1.50063188 1.52961942 1.55779029  
1.58518927 1.61185752 1.637833 1.66315081 1.68784342 1.71194097  
1.73547147 1.75846099 1.78093385 1.80291275 1.82441896 1.84547237  
1.86609165 1.88629436 1.90609699 1.92551507 1.94456327 1.9632554  
1.98160454 1.99962305 2.01732262 2.03471437 2.0518088 2.06861592  
2.08514522 2.10140574 2.11740608 2.13315444 2.14865863 2.1639261  
2.17896398 2.19377906 2.20837786 2.2227666 2.23695123 2.25093747  
2.2647308 2.27833645 2.29175947 2.3050047 2.31807678 2.33098018  
2.34371921 2.35629799 2.36872051 2.3809906 2.39311196 2.40508815  
2.41692261 2.42861865 2.44017947 2.45160817 2.46290773 2.47408103  
2.48513086 2.49605993 2.50687085 2.51756614 2.52814825 2.53861955  
2.54898233 2.55923883 2.56939121 2.57944154 2.58939187 2.59924417  
2.60900034 2.61866225 2.62823171 2.63771045 2.64710019 2.65640258  
2.66561924 2.67475172 2.68380156 2.69277023 2.70165917 2.7104698  
2.71920348 2.72786155 2.73644529 2.74495598 2.75339485 2.7617631  
2.7700619 2.7782924 2.78645571 2.79455292], shape=(100,), dtype=float64)  
tf.Tensor(  
[0.75 0.72115385 0.69444444 0.66964286 0.64655172 0.625  
0.60483871 0.5859375 0.56818182 0.55147059 0.53571429 0.52083333  
0.50675676 0.49342105 0.48076923 0.46875 0.45731707 0.44642857  
0.43604651 0.42613636 0.41666667 0.4076087 0.39893617 0.390625  
0.38265306 0.375 0.36764706 0.36057692 0.35377358 0.34722222  
0.34090909 0.33482143 0.32894737 0.32327586 0.31779661 0.3125  
0.30737705 0.30241935 0.29761905 0.29296875 0.28846154 0.28409091  
0.27985075 0.27573529 0.27173913 0.26785714 0.26408451 0.26041667  
0.25684932 0.25337838 0.25 0.24671053 0.24350649 0.24038462  
0.23734177 0.234375 0.23148148 0.22865854 0.22590361 0.22321429  
0.22058824 0.21802326 0.21551724 0.21306818 0.21067416 0.20833333  
0.20604396 0.20380435 0.2016129 0.19946809 0.19736842 0.1953125  
0.19329897 0.19132653 0.18939394 0.1875 0.18564356 0.18382353  
0.18203883 0.18028846 0.17857143 0.17688679 0.17523364 0.17361111  
0.17201835 0.17045455 0.16891892 0.16741071 0.1659292 0.16447368  
0.16304348 0.16163793 0.16025641 0.15889831 0.15756303 0.15625  
0.15495868 0.15368852 0.15243902 0.15120968], shape=(100,), dtype=float64)  
tf.Tensor(  
[-0.25 -0.23113905 -0.21433471 -0.19929847 -0.18579073 -0.17361111  
-0.16259105 -0.15258789 -0.14348026 -0.13516436 -0.12755102 -0.12056327  
-0.1141344 -0.10820637 -0.10272847 -0.09765625 -0.09295062 -0.0885771  
-0.08450514 -0.08070764 -0.07716049 -0.07384216 -0.07073336 -0.06781684  
-0.06507705 -0.0625 -0.06007305 -0.05778476 -0.05562478 -0.05358368  
-0.05165289 -0.04982462 -0.04809172 -0.04644768 -0.04488653 -0.04340278  
-0.0419914 -0.04064776 -0.0393676 -0.03814697 -0.03698225 -0.03587006  
-0.03480731 -0.03379109 -0.03281874 -0.03188776 -0.03099583 -0.03014082  
-0.0293207 -0.0285336 -0.02777778 -0.02705159 -0.02635352 -0.02568212  
-0.02503605 -0.02441406 -0.02381497 -0.02323766 -0.02268109 -0.02214427  
-0.0216263 -0.02112628 -0.02064341 -0.02017691 -0.01972604 -0.01929012  
-0.01886849 -0.01846054 -0.01806567 -0.01768334 -0.01731302 -0.01695421  
-0.01660644 -0.01626926 -0.01594225 -0.015625 -0.01531713 -0.01501826  
-0.01472806 -0.01444619 -0.01417234 -0.01390619 -0.01364748 -0.01339592  
-0.01315125 -0.01291322 -0.0126816 -0.01245615 -0.01223667 -0.01202293  
-0.01181474 -0.01161192 -0.01141427 -0.01122163 -0.01103383 -0.01085069  
-0.01067209 -0.01049785 -0.01032785 -0.01016194], shape=(100,), dtype=float64)
```

```
In [200... X_mesh, Y_mesh = np.meshgrid(x, y)
# Z_mesh = np.zeros((100, 1)) + dxxdy.numpy()
Z_mesh = np.tile(dxxdy, (dxxdy.shape[0], 1))
print(X_mesh.shape, Y_mesh.shape, dxxdy.shape[0])
# Z_mesh = np.meshgrid(z, z)
X_mesh, Y_mesh, Z_mesh
```

```
(100, 100) (100, 100) 100
```

```
Out[200... (array([[1. , 1.04, 1.08, ..., 4.88, 4.92, 4.96],
        [1. , 1.04, 1.08, ..., 4.88, 4.92, 4.96],
        [1. , 1.04, 1.08, ..., 4.88, 4.92, 4.96],
        ...,
        [1. , 1.04, 1.08, ..., 4.88, 4.92, 4.96],
        [1. , 1.04, 1.08, ..., 4.88, 4.92, 4.96],
        [1. , 1.04, 1.08, ..., 4.88, 4.92, 4.96]]),
array([[1. , 1. , 1. , ..., 1. , 1. , 1. ],
        [1.04, 1.04, 1.04, ..., 1.04, 1.04, 1.04],
        [1.08, 1.08, 1.08, ..., 1.08, 1.08, 1.08],
        ...,
        [4.88, 4.88, 4.88, ..., 4.88, 4.88, 4.88],
        [4.92, 4.92, 4.92, ..., 4.92, 4.92, 4.92],
        [4.96, 4.96, 4.96, ..., 4.96, 4.96, 4.96]]),
array([[ -0.25      , -0.23113905, -0.21433471, ..., -0.01049785,
        -0.01032785, -0.01016194],
        [ -0.25      , -0.23113905, -0.21433471, ..., -0.01049785,
        -0.01032785, -0.01016194],
        [ -0.25      , -0.23113905, -0.21433471, ..., -0.01049785,
        -0.01032785, -0.01016194],
        ...,
        [ -0.25      , -0.23113905, -0.21433471, ..., -0.01049785,
        -0.01032785, -0.01016194],
        [ -0.25      , -0.23113905, -0.21433471, ..., -0.01049785,
        -0.01032785, -0.01016194],
        [ -0.25      , -0.23113905, -0.21433471, ..., -0.01049785,
        -0.01032785, -0.01016194]]))
```

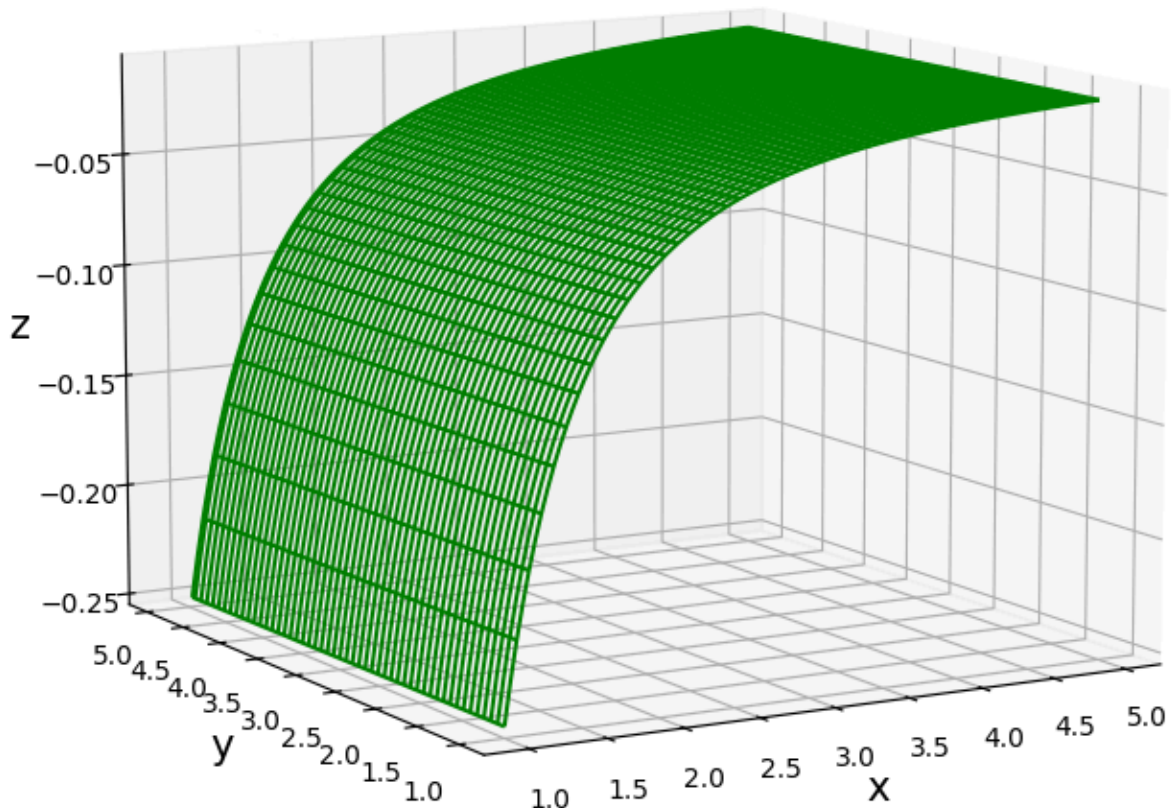
```
In [201... fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# def f(x, y):
#     return x * tf.math.log(x+y)
ax.plot_wireframe(X_mesh, Y_mesh, Z_mesh, color = 'green')

ax.view_init(azim = -120, elev = 10)
ax.set_title("d3/dx2dy(x*ln(x+y))")
ax.set_xlabel('x', fontsize = 15)
ax.set_ylabel('y', fontsize = 15)
ax.set_zlabel('z', fontsize = 15)
```

```
Out[201... Text(0.5, 0, 'z')
```

$$d^3/dx^2dy(x*\ln(x+y))$$



7. Решите задачу парной линейной регрессии при помощи модели TensorFlow, рассматривая тензор ранга 1 из пункта 1 задания как значения зависимой переменной (отклика), а точки отрезка из индивидуального задания как значения независимой переменной (предиктора). Предварительно масштабируйте независимую и зависимую переменные на интервал  $[0, 1]$ . Оцените качество полученной модели по показателю качества регрессии, указанному в индивидуальном задании. Количество эпох, скорость обучения и

## начальные значения весов выберите самостоятельно, обеспечивая сходимость обучения.

Показатель качества регрессии -- MaxErr

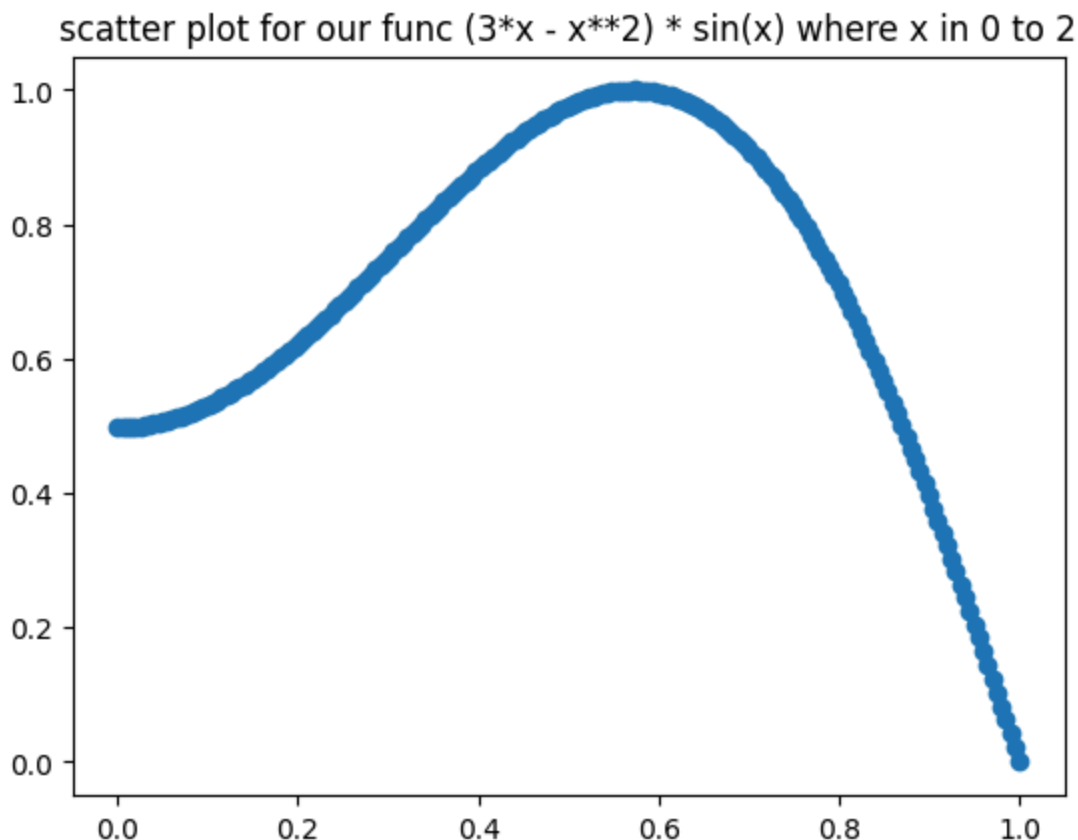
```
In [330...] X, Y = tensor_xy[0], tensor_xy[1]
xn = (X - tf.reduce_min(X))/(tf.reduce_max(X) - tf.reduce_min(X))
yn = (Y - tf.reduce_min(Y))/(tf.reduce_max(Y) - tf.reduce_min(Y))

xn[:5], yn[:5]
```

```
Out[330...] (<tf.Tensor: shape=(5,), dtype=float32, numpy=
  array([0.          , 0.00502513, 0.01005025, 0.01507538, 0.0201005 ],
        dtype=float32)>,
  <tf.Tensor: shape=(5,), dtype=float32, numpy=
  array([0.49769297, 0.4977785 , 0.4980351 , 0.49846262, 0.49906075],
        dtype=float32)>)
```

```
In [331...] plt.scatter(xn, yn)
plt.title('scatter plot for our func (3*x - x**2) * sin(x) where x in 0 to 2')
```

```
Out[331...] Text(0.5, 1.0, 'scatter plot for our func (3*x - x**2) * sin(x) where x in 0 to 2')
```






















```
In [332...] reg = tf.keras.Sequential([
  tf.keras.Input(shape=(1,)),
```

```
tf.keras.layers.Dense(units=1, kernel_regularizer = tf.keras.regularizers.L1L2(1))
```



















```
In [333... def MaxErr(y_pred, y_real):  
    accuracy = tf.reduce_max(y_real - y_pred)  
    return accuracy
```

```
In [334... reg.compile(  
    optimizer=tf.optimizers.Adam(learning_rate=0.01),  
    loss='mean_absolute_error',  
    metrics = [MaxErr]  
)
```

```
In [335... uhist = reg.fit(  
    xn, yn,  
    epochs = 50,  
    verbose = 1,  
    validation_split = 0.3  
)
```

Epoch 1/50  
5/5  2s 94ms/step - loss: 0.6030 - max\_err: -0.1452 - val\_loss: 0.2862 - val\_max\_err: 0.2893  
Epoch 2/50  
5/5  0s 14ms/step - loss: 0.5367 - max\_err: -0.0633 - val\_loss: 0.2912 - val\_max\_err: 0.3853  
Epoch 3/50  
5/5  0s 18ms/step - loss: 0.4661 - max\_err: 0.0257 - val\_loss: 0.3116 - val\_max\_err: 0.4815  
Epoch 4/50  
5/5  0s 17ms/step - loss: 0.4019 - max\_err: 0.1101 - val\_loss: 0.3481 - val\_max\_err: 0.5778  
Epoch 5/50  
5/5  0s 18ms/step - loss: 0.3367 - max\_err: 0.1926 - val\_loss: 0.4015 - val\_max\_err: 0.6744  
Epoch 6/50  
5/5  0s 20ms/step - loss: 0.2684 - max\_err: 0.2737 - val\_loss: 0.4724 - val\_max\_err: 0.7702  
Epoch 7/50  
5/5  0s 23ms/step - loss: 0.2070 - max\_err: 0.3451 - val\_loss: 0.5608 - val\_max\_err: 0.8653  
Epoch 8/50  
5/5  0s 18ms/step - loss: 0.1389 - max\_err: 0.4379 - val\_loss: 0.6524 - val\_max\_err: 0.9594  
Epoch 9/50  
5/5  0s 16ms/step - loss: 0.0880 - max\_err: 0.5318 - val\_loss: 0.7376 - val\_max\_err: 1.0467  
Epoch 10/50  
5/5  0s 17ms/step - loss: 0.0543 - max\_err: 0.5977 - val\_loss: 0.7970 - val\_max\_err: 1.1075  
Epoch 11/50  
5/5  0s 16ms/step - loss: 0.0635 - max\_err: 0.6413 - val\_loss: 0.8149 - val\_max\_err: 1.1258  
Epoch 12/50  
5/5  0s 16ms/step - loss: 0.0696 - max\_err: 0.6556 - val\_loss: 0.7965 - val\_max\_err: 1.1068  
Epoch 13/50  
5/5  0s 16ms/step - loss: 0.0641 - max\_err: 0.6270 - val\_loss: 0.7693 - val\_max\_err: 1.0787  
Epoch 14/50  
5/5  0s 17ms/step - loss: 0.0590 - max\_err: 0.6152 - val\_loss: 0.7475 - val\_max\_err: 1.0564  
Epoch 15/50  
5/5  0s 13ms/step - loss: 0.0557 - max\_err: 0.5829 - val\_loss: 0.7369 - val\_max\_err: 1.0454  
Epoch 16/50  
5/5  0s 14ms/step - loss: 0.0584 - max\_err: 0.5875 - val\_loss: 0.7355 - val\_max\_err: 1.0439  
Epoch 17/50  
5/5  0s 16ms/step - loss: 0.0538 - max\_err: 0.5813 - val\_loss: 0.7387 - val\_max\_err: 1.0471  
Epoch 18/50  
5/5  0s 12ms/step - loss: 0.0582 - max\_err: 0.5891 - val\_loss: 0.7448 - val\_max\_err: 1.0533  
Epoch 19/50  
5/5  0s 15ms/step - loss: 0.0543 - max\_err: 0.5931 - val\_loss:



0.7467 - val\_max\_err: 1.0552  
Epoch 20/50  
5/5  0s 15ms/step - loss: 0.0557 - max\_err: 0.5948 - val\_loss: 0.7472 - val\_max\_err: 1.0558  
Epoch 21/50  
5/5  0s 13ms/step - loss: 0.0542 - max\_err: 0.5834 - val\_loss: 0.7428 - val\_max\_err: 1.0512  
Epoch 22/50  
5/5  0s 14ms/step - loss: 0.0582 - max\_err: 0.5960 - val\_loss: 0.7383 - val\_max\_err: 1.0465  
Epoch 23/50  
5/5  0s 14ms/step - loss: 0.0561 - max\_err: 0.5868 - val\_loss: 0.7373 - val\_max\_err: 1.0454  
Epoch 24/50  
5/5  0s 12ms/step - loss: 0.0547 - max\_err: 0.5867 - val\_loss: 0.7387 - val\_max\_err: 1.0469  
Epoch 25/50  
5/5  0s 12ms/step - loss: 0.0558 - max\_err: 0.5926 - val\_loss: 0.7375 - val\_max\_err: 1.0456  
Epoch 26/50  
5/5  0s 16ms/step - loss: 0.0557 - max\_err: 0.5947 - val\_loss: 0.7363 - val\_max\_err: 1.0444  
Epoch 27/50  
5/5  0s 12ms/step - loss: 0.0558 - max\_err: 0.5874 - val\_loss: 0.7340 - val\_max\_err: 1.0420  
Epoch 28/50  
5/5  0s 14ms/step - loss: 0.0569 - max\_err: 0.5841 - val\_loss: 0.7373 - val\_max\_err: 1.0453  
Epoch 29/50  
5/5  0s 12ms/step - loss: 0.0587 - max\_err: 0.5771 - val\_loss: 0.7416 - val\_max\_err: 1.0497  
Epoch 30/50  
5/5  0s 11ms/step - loss: 0.0555 - max\_err: 0.5810 - val\_loss: 0.7450 - val\_max\_err: 1.0532  
Epoch 31/50  
5/5  0s 31ms/step - loss: 0.0576 - max\_err: 0.5904 - val\_loss: 0.7479 - val\_max\_err: 1.0563  
Epoch 32/50  
5/5  0s 16ms/step - loss: 0.0574 - max\_err: 0.5868 - val\_loss: 0.7455 - val\_max\_err: 1.0539  
Epoch 33/50  
5/5  0s 20ms/step - loss: 0.0552 - max\_err: 0.5892 - val\_loss: 0.7449 - val\_max\_err: 1.0533  
Epoch 34/50  
5/5  0s 16ms/step - loss: 0.0535 - max\_err: 0.5821 - val\_loss: 0.7432 - val\_max\_err: 1.0516  
Epoch 35/50  
5/5  0s 22ms/step - loss: 0.0546 - max\_err: 0.5950 - val\_loss: 0.7366 - val\_max\_err: 1.0448  
Epoch 36/50  
5/5  0s 20ms/step - loss: 0.0567 - max\_err: 0.5858 - val\_loss: 0.7371 - val\_max\_err: 1.0452  
Epoch 37/50  
5/5  0s 22ms/step - loss: 0.0594 - max\_err: 0.5924 - val\_loss: 0.7397 - val\_max\_err: 1.0479  
Epoch 38/50

```

5/5 ————— 0s 26ms/step - loss: 0.0540 - max_err: 0.5646 - val_loss:
0.7411 - val_max_err: 1.0494
Epoch 39/50
5/5 ————— 0s 14ms/step - loss: 0.0592 - max_err: 0.5875 - val_loss:
0.7461 - val_max_err: 1.0546
Epoch 40/50
5/5 ————— 0s 12ms/step - loss: 0.0543 - max_err: 0.5907 - val_loss:
0.7474 - val_max_err: 1.0559
Epoch 41/50
5/5 ————— 0s 16ms/step - loss: 0.0547 - max_err: 0.5928 - val_loss:
0.7471 - val_max_err: 1.0556
Epoch 42/50
5/5 ————— 0s 20ms/step - loss: 0.0550 - max_err: 0.6035 - val_loss:
0.7435 - val_max_err: 1.0519
Epoch 43/50
5/5 ————— 0s 16ms/step - loss: 0.0548 - max_err: 0.5758 - val_loss:
0.7403 - val_max_err: 1.0485
Epoch 44/50
5/5 ————— 0s 16ms/step - loss: 0.0583 - max_err: 0.5891 - val_loss:
0.7372 - val_max_err: 1.0454
Epoch 45/50
5/5 ————— 0s 14ms/step - loss: 0.0572 - max_err: 0.5895 - val_loss:
0.7419 - val_max_err: 1.0501
Epoch 46/50
5/5 ————— 0s 17ms/step - loss: 0.0547 - max_err: 0.5966 - val_loss:
0.7423 - val_max_err: 1.0505
Epoch 47/50
5/5 ————— 0s 12ms/step - loss: 0.0564 - max_err: 0.5848 - val_loss:
0.7443 - val_max_err: 1.0526
Epoch 48/50
5/5 ————— 0s 11ms/step - loss: 0.0564 - max_err: 0.5898 - val_loss:
0.7447 - val_max_err: 1.0529
Epoch 49/50
5/5 ————— 0s 14ms/step - loss: 0.0558 - max_err: 0.5946 - val_loss:
0.7442 - val_max_err: 1.0525
Epoch 50/50
5/5 ————— 0s 16ms/step - loss: 0.0597 - max_err: 0.5775 - val_loss:
0.7341 - val_max_err: 1.0421

```

**8. Постройте кривую обучения для показателя качества регрессии, указанного в индивидуальном задании, с зависимостью от количества эпох. Показатель качества регрессия реализуйте как функцию с использованием функций модуля `tf.math`.**

In [336...

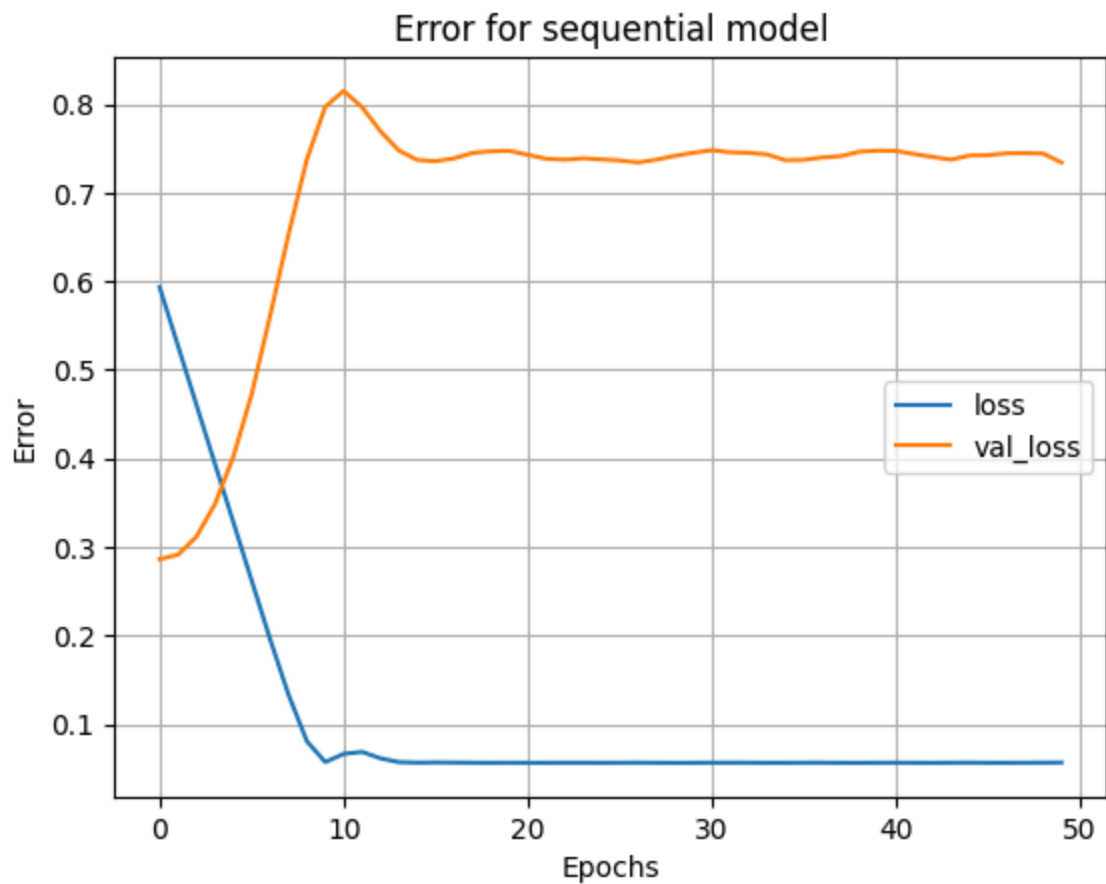
```

def plot_loss(history):
    plt.plot(history.history['loss'], label = 'loss')
    plt.plot(history.history['val_loss'], label = 'val_loss')
    # plt.ylim([0, max(history.history['loss'])])
    plt.xlabel('Epochs')
    plt.ylabel('Error')

```

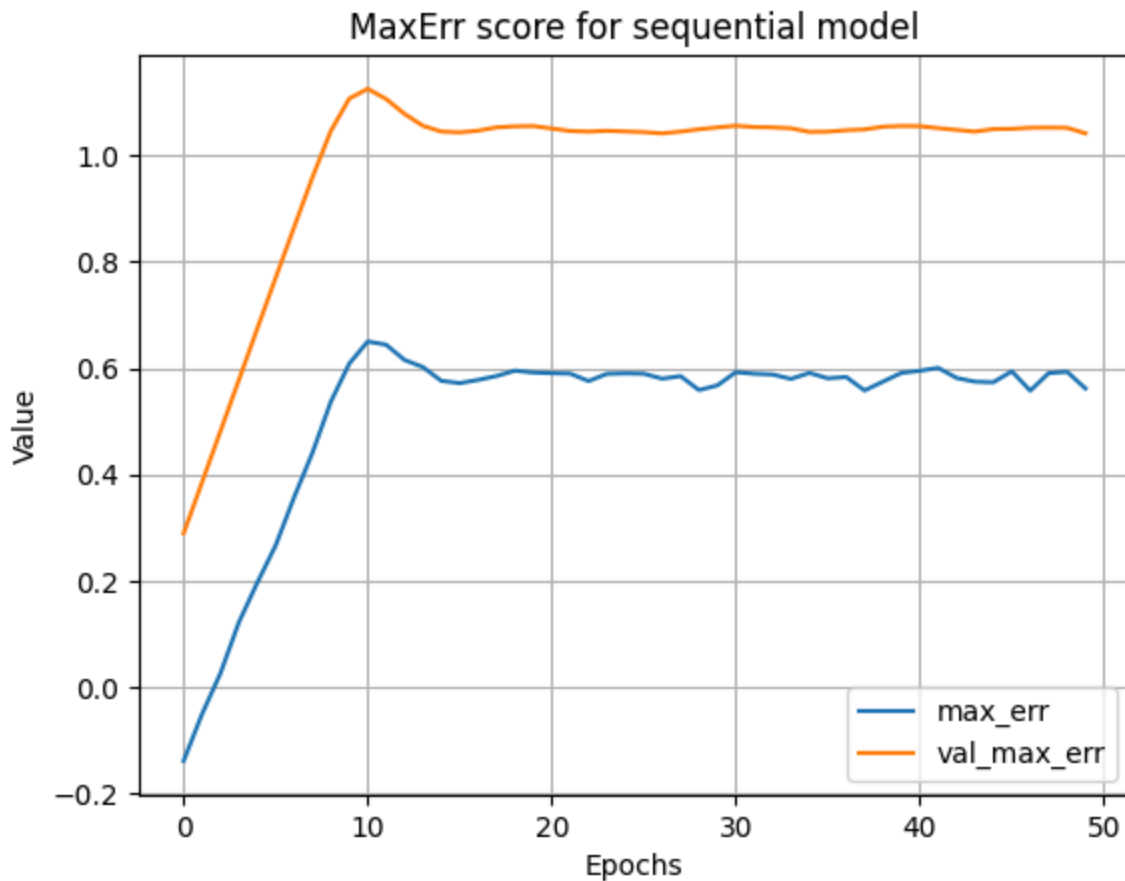
```
plt.title('Error for sequential model')  
plt.legend()  
plt.grid(True)
```

In [337... `plot_loss(uhist)`



```
In [338... def plot_metrics(history):  
    plt.plot(history.history['max_err'], label = 'max_err')  
    plt.plot(history.history['val_max_err'], label = 'val_max_err')  
    # plt.ylim([0, max(history.history['r2_score'])*1])  
    plt.xlabel('Epochs')  
    plt.ylabel('Value')  
    plt.title('MaxErr score for sequential model')  
    plt.legend()  
    plt.grid(True)
```

In [339... `plot_metrics(uhist)`



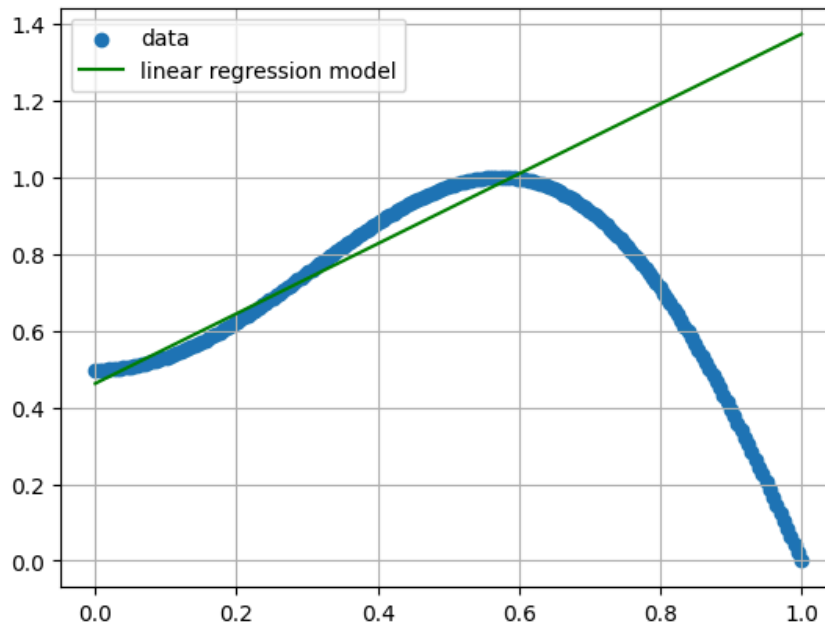
9. Изобразите на графике точки набора данных (независимой и зависимой переменных) и линию построенной парной регрессии, подписывая оси и рисунок и создавая легенду.

In [340... `y_pred = reg.predict(xn)`

7/7 ————— 0s 10ms/step

In [341... `plt.scatter(xn, yn, label = 'data')`  
`plt.plot(xn, y_pred, label = 'linear regression model', c = 'green')`  
`plt.title('min-maxed func (3*x - x^2) * sin(x), where x was in 0 to 2, and it\'s li`  
`plt.legend()`  
`plt.grid(True)`

min-maxed func  $(3x - x^2) * \sin(x)$ , where  $x$  was in 0 to 2, and it's linear regression model



Как мы видим, к сожалению, у нас не получится идеально решить задачу моделью линейной регрессии, так как изначальная функция похожа на синусоиду, из-за чего периодически она будет убывать и снова возрастать.

In [ ]: