Prof. Dr. L. Paditz, 14.02.2019

HTW Dresden,

paditz@htw-dresden.de

Mathematik-Grundlagen als eActivty mit ClassPad

lineare Regression in TensorFlow

Gradientenverfahren - steilster Abstieg zum Min!

==========

Quelle:

https://github.com/aymericdamien/TensorFlow-Examples/

blob/master/examples/2_BasicModels/linear_regression.py

s. auch:

https://github.com/tensorflow/tensorflow/issues/8630

oder:

https://stackoverflow.com/questions/43170017/

linear-regression-with-tensorflow

https://stackoverflow.com/questions/43170017/

linear-regression-with-tensorflow/43170585

und

https://www.geeksforgeeks.org/introduction-to-tensorflo

und

http://www.machineintellegence.com/linearregression/

Suchworte in google.de:

tensorflow "Optimization Finished!" "Training cost= ["

"[3.3,4.4,5.5,6.71,6.93,"

train_X:={3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.

{3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.5

train_Y:={1.7,2.76,2.09,3.19,1.694,1.573,3.366

{1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,

STAT-Menü lin.Regression

DelVar W,b

done

X_liste:=listToMat(train_X)

$$\begin{bmatrix} 3.3 \\ 4.4 \\ 5.5 \\ 6.71 \\ 6.93 \\ 4.168 \\ 9.779 \\ 6.182 \\ 7.59 \\ 2.167 \\ 7.042 \\ 10.791 \\ 5.313 \\ 7.997 \\ 5.654 \\ 9.27 \\ 3.1 \end{bmatrix}$$

Y_liste:=listToMat(train_Y)

$$\begin{bmatrix} 1.7 \\ 2.76 \\ 2.09 \\ 3.19 \\ 1.694 \\ 1.573 \\ 3.366 \\ 2.596 \\ 2.53 \\ 1.221 \\ 2.827 \\ 3.465 \\ 1.65 \\ 2.904 \\ 2.42 \\ 2.94 \\ 1.3 \end{bmatrix}$$

Differenzvektor:

D_liste:=Y_liste−W∗X_liste−b∗fill(1,17,1)

$$\begin{bmatrix} -3.3\cdot W-b+1.7 \\ -4.4\cdot W-b+2.76 \\ -5.5\cdot W-b+2.09 \\ -6.71\cdot W-b+3.19 \\ -6.93\cdot W-b+1.694 \\ -4.168\cdot W-b+1.573 \\ -9.779\cdot W-b+3.366 \\ -6.182\cdot W-b+2.596 \\ -7.59\cdot W-b+2.53 \\ -2.167\cdot W-b+1.221 \\ -7.042\cdot W-b+2.827 \\ -10.791\cdot W-b+3.465 \\ -5.313\cdot W-b+1.65 \\ -7.997\cdot W-b+2.904 \\ -5.654\cdot W-b+2.42 \\ -9.27\cdot W-b+2.94 \\ -3.1\cdot W-b+1.3 \end{bmatrix}$$

algebraische Lösung: Orthogonalitätsbedingungen:

dotP(D_liste,X_liste)=0

$$-2.167\cdot(2.167\cdot W+b-1.221)-3.1\cdot(3.1\cdot W+b-1.3)-4.$$

Equ1:=expand(ans)

$$-752.797217\cdot W-105.893\cdot b+274.017544=0$$

dotP(D_liste,fill(1,17,1))=0

$$-105.893\cdot W-17\cdot b+40.226=0$$

Equ2:=ans

$$-105.893\cdot W-17\cdot b+40.226=0$$

$$\left\{\begin{matrix}\text{Equ1}\\ \text{Equ2}\end{matrix}\right|_{W,b}$$

$$\{W=0.2516349443, b=0.7988012262\}$$

Lineare Regression:

==========

LinearReg train_X,train_Y,1,y10

done

DispStat

done

y10(x)

0.2516349443·x+0.7988012262

stop

==========

Lineare Regression

y=a*x+b

a = 0.2516349

b = 0.7988012

r = 0.8323918

r^2 = 0.692876

MSe = 0.174372

==========

analytische Lösung: grad(cost(W,b)=$\begin{bmatrix}0\\0\end{bmatrix}$

dotP(D_liste,D_liste)/dim(train_X)

0.05882352941·((10.791·W+b−3.465)^2+(9.779·W+▶

expand(ans)

44.28218924·W^2+b^2+12.458·W·b−32.23735812·W−4▶

Define cost(W,b)=44.28218924·W^2+b^2+12.458·W·b−▶

done

$$\left\{\begin{array}{l}\frac{d}{dW}(cost(W,b))=0\\ \frac{d}{db}(cost(W,b))=0\end{array}\right|_{W,b}$$

{W=0.2516349444,b=0.7988012252}

Define z1(x,y)=cost(x,y)

done

stop

3D-Grafik Z1:… Z2:…

solve(s=cost(W,b),b)

{b=-3.223775793E-28·(1.932206332E+28·W-941429 ▶

expand(ans)

{b=-6.229·W+3.034957485E-19·(-5.951327671E+37 ▶

getRight(ans)

{-6.229·W+1.720568677E-25·(-1.763926965E+24·(▶

listToMat(ans)

$$\left[\begin{array}{l}-6.229\cdot W+1.720568677E-25\cdot(-1.763926965E+24\cdot(\\ -6.229\cdot W-1.720568677E-25\cdot(-1.763926965E+24\cdot(\end{array}\right]$$

ans[2,1]

-6.229·W-1.720568677E-25·(-1.763926965E+24·(1. ▶

-6.229·W-(-1.763926965E+24·(1.720568677E-25)^2 ▶

-6.229·W-(-5.221852784E-26·(1.049770736E+26·W ▶

-6.229·W-(-(1.049770736E+26·5.221852784E-26·W ▶

-6.229·W-(-5.48174824·W^2+2.758798826·W+0.99 ▶

Definition von Höhenlinien der cost-Funktion:
x=t, y=W, z=cost(W,b)

Define xst2(s,t)=t

done

Define yst2(s,t)=-6.229·t-(-5.48174824·t^2+2.7587 ▶

done

Define zst2(s,t)=s

done

Define xst3(s,t)=t

done

Define yst3(s,t)=-6.229·t+(-5.48174824·t^2+2.7587 ▶

done

Define zst3(s,t)=s

done

stop

3D-Grafik Z1:··· Z2:···

2D-Grafik Höhenlinien Y1:··· Y2:···

1.Iteration mit tensorflow:

==========

Startwerte per Zufall ausgewählt: W= [0.38903466]
b= [-0.40474084]
Epoch: 0000 cost= 0.177051708 W= [0.39435494]
b= [-0.3878391]

```
Epoch: 0001 cost= 0.174632162 W= [0.39232808]
b= [-0.37236074]
Epoch: 0002 cost= 0.172295541 W= [0.39030755]
b= [-0.35708445]
Epoch: 0003 cost= 0.170018047 W= [0.38831297]
b= [-0.34200448]
Epoch: 0004 cost= 0.167798087 W= [0.38634402]
b= [-0.32711828]
Epoch: 0005 cost= 0.165634304 W= [0.38440034]
b= [-0.31242332]
Epoch: 0006 cost= 0.163525149 W= [0.38248163]
b= [-0.29791716]
Epoch: 0007 cost= 0.161469370 W= [0.38058755]
b= [-0.2835974]
Epoch: 0008 cost= 0.159465447 W= [0.37871787]
b= [-0.2694616]
Epoch: 0009 cost= 0.157512143 W= [0.37687218]
b= [-0.25550747]

W:={0.38903466,0.39435494,0.39232808,0.39030▶
    {0.38903466,0.39435494,0.39232808,0.39030755▶
dim(ans)
                                                  11
b:={-0.40474084,-0.3878391,-0.37236074,-0.357▶
    {-0.40474084,-0.3878391,-0.37236074,-0.35708▶
dim(ans)
                                                  11
```

2. Iteration mit tensorflow:

==========

```
Startwerte per Zufall ausgewählt:  W= [-0.6830545]
b= [1.1212226]
Epoch: 0000 cost= 0.093539596 W= [0.17438477]
b= [1.257438]
Epoch: 0010 cost= 0.089693040 W= [0.18378624]
b= [1.2042997]
Epoch: 0020 cost= 0.087661669 W= [0.1900068] b=
[1.1572697]
Epoch: 0030 cost= 0.086108372 W= [0.19547294]
b= [1.1159436]
Epoch: 0040 cost= 0.084922418 W= [0.20027587]
b= [1.0796316]
Epoch: 0050 cost= 0.084018461 W= [0.20449626]
b= [1.0477235]
Epoch: 0060 cost= 0.083330818 W= [0.2082046] b=
[1.019687]
Epoch: 0070 cost= 0.082808919 W= [0.21146306]
b= [0.99505186]
Epoch: 0080 cost= 0.082413912 W= [0.21432634]
b= [0.9734042]
Epoch: 0090 cost= 0.082115903 W= [0.21684231]
b= [0.9543826]
Optimization Finished!
Training cost= [0.08191158] W= [0.21884456] b=
[0.9392446]

W1:={-0.6830545,0.17438477,0.18378624,0.1900▶
     {-0.6830545,0.17438477,0.18378624,0.1900068,▶
b1:={1.1212226,1.257438,1.2042997,1.1572697,1▶
```

{1.1212226,1.257438,1.2042997,1.1572697,1.11

dim(W1)=dim(b1)

11=11

3.Iteration mit tensorflow:

==========

Startwerte per Zufall ausgewählt: W= [0.16528997]
b= [-0.08805541]
Epoch: 0000 cost= 0.529334068 W= [0.22420149]
b= [0.020639995]
Epoch: 0001 cost= 0.500386834 W= [0.21591327]
b= [0.10483948]
Epoch: 0002 cost= 0.475245655 W= [0.20835607]
b= [0.1816125]
Epoch: 0003 cost= 0.453363746 W= [0.2014654] b=
[0.25161412]
Epoch: 0004 cost= 0.434277445 W= [0.19518256]
b= [0.31544152]
Epoch: 0005 cost= 0.417595029 W= [0.18945378]
b= [0.37363935]
Epoch: 0006 cost= 0.402982056 W= [0.18423039]
b= [0.42670402]
Epoch: 0007 cost= 0.390155882 W= [0.17946766]
b= [0.47508836]
Epoch: 0008 cost= 0.378874570 W= [0.17512502]
b= [0.5192052]
Epoch: 0009 cost= 0.368932009 W= [0.17116545]
b= [0.55943096]
Optimization Finished!

Training cost= [0.368932] W= [0.17116545] b=
[0.55943096]

W2:={0.16528997,0.22420149,0.21591327,0.2083▶
{0.16528997,0.22420149,0.21591327,0.20835607▶
b2:={-0.08805541,0.020639995,0.10483948,0.181▶
{-0.08805541,0.020639995,0.10483948,0.181612▶
dim(W2)=dim(b2)

11=11

4. Iteration mit tensorflow:

==========

Startwerte per Zufall ausgewählt: W= [-0.9822901]
b= [-0.32099992]
Epoch: 0000 cost= 0.515979826 W= [0.22043358]
b= [0.058935374]
Epoch: 0010 cost= 0.356591940 W= [0.16605784]
b= [0.6113189]
Epoch: 0020 cost= 0.310448349 W= [0.14446126]
b= [0.8307175]
Epoch: 0030 cost= 0.294828981 W= [0.13588344]
b= [0.91785926]
Epoch: 0040 cost= 0.289052278 W= [0.13247648]
b= [0.95247066]
Epoch: 0050 cost= 0.286825418 W= [0.13112327]
b= [0.9662177]
Epoch: 0060 cost= 0.285951734 W= [0.13058577]
b= [0.9716777]
Epoch: 0070 cost= 0.285606116 W= [0.13037232]

b= [0.9738466]
Epoch: 0080 cost= 0.285469115 W= [0.13028754]
b= [0.974708]
Epoch: 0090 cost= 0.285414904 W= [0.13025387]
b= [0.9750499]
Optimization Finished!
Training cost= [0.28539455] W= [0.13024136] b=
[0.97517717]

W3:={-0.9822901,0.22043358,0.16605784,0.1444▶
{-0.9822901,0.22043358,0.16605784,0.14446126▶
b3:={-0.32099992,0.058935374,0.6113189,0.8307▶
{-0.32099992,0.058935374,0.6113189,0.8307175▶
dim(W3)=dim(b3)

12=12

STAT-Menü 

Skript:

python3
'''
A linear regression learning algorithm example using
TensorFlow library.
Author: Aymeric Damien
Project:
https://github.com/aymericdamien/TensorFlow-Examples/
'''

from __future__ import print_function

```
import tensorflow as tf
from numpy import *
import numpy
import matplotlib.pyplot as plt
rng = numpy.random

# Parameters
learning_rate = 0.0001
training_epochs = 1000
display_step = 50

# Training Data
train_X =
numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.7

7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y =
numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,

2.827,3.465,1.65,2.904,2.42,2.94,1.3])

train_X=numpy.asarray(train_X)
train_Y=numpy.asarray(train_Y)
n_samples = train_X.shape[0]


# tf Graph Input
X = tf.placeholder("float")
```

```
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")

# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)



# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y,
2))/(2*n_samples)
# Gradient descent
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi▶

# Initializing the variables
init = tf.global_variables_initializer()

# Launch the graph
with tf.Session() as sess:
    sess.run(init)
    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y:
y})
        # Display logs per epoch step
```

```
            if (epoch+1) % display_step == 0:
                c = sess.run(cost, feed_dict={X:
train_X, Y:train_Y})
                print("Epoch:", '%04d' % (epoch+1),
"cost=", "{:.9f}".format(c), \
                    "W=", sess.run(W), "b=",
sess.run(b))
    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict={X:
train_X, Y: train_Y})
    print("Training cost=", training_cost, "W=",
sess.run(W), "b=", sess.run(b), '\n')
    # Graphic display
    plt.plot(train_X, train_Y, 'ro', label='Original
data')
    plt.plot(train_X, sess.run(W) * train_X +
sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()
```

==========

Rechnerprotokoll:

```
parallels@parallels-Parallels-Virtual-Platform:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> '''
```

```
... A linear regression learning algorithm example using
TensorFlow library.
... Author
... Project:
https://github.com/aymericdamien/TensorFlow-Examples/
... '''
'\nA linear regression learning algorithm example using
TensorFlow library.\nAuthor\nProject:
https://github.com/aymericdamien/TensorFlow-Examples
>>>
>>> from __future__ import print_function
>>>
>>> import tensorflow as tf
>>> from numpy import *
>>> import numpy
>>> import matplotlib.pyplot as plt
>>> rng = numpy.random
>>>
>>> # Parameters
... learning_rate = 0.0001
>>> training_epochs = 1000
>>> display_step = 50
>>>
>>> # Training Data
... train_X =
numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.7
...
7.042,10.791,5.313,7.997,5.654,9.27,3.1])
>>> train_Y =
```

```
numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,
...
2.827,3.465,1.65,2.904,2.42,2.94,1.3])
>>>
>>> train_X=numpy.asarray(train_X)
>>> train_Y=numpy.asarray(train_Y)
>>> n_samples = train_X.shape[0]
>>>
>>>
>>> # tf Graph Input
... X = tf.placeholder("float")
>>> Y = tf.placeholder("float")
>>>
>>> # Set model weights
... W = tf.Variable(rng.randn(), name="weight")
>>> b = tf.Variable(rng.randn(), name="bias")
>>>
>>> # Construct a linear model
... pred = tf.add(tf.multiply(X, W), b)
>>>
>>>
>>> # Mean squared error
... cost = tf.reduce_sum(tf.pow(pred-Y,
2))/(2*n_samples)
>>> # Gradient descent
... optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimi
>>>
>>> # Initializing the variables
```

```
... init = tf.global_variables_initializer()
>>>
>>> # Launch the graph
... with tf.Session() as sess:
...     sess.run(init)
...     # Fit all training data
...     for epoch in range(training_epochs):
...         for (x, y) in zip(train_X, train_Y):
...             sess.run(optimizer, feed_dict={X:
x, Y: y})
...         # Display logs per epoch step
...         if (epoch+1) % display_step == 0:
...             c = sess.run(cost, feed_dict={X:
train_X, Y:train_Y})
...             print("Epoch:", '%04d' %
(epoch+1), "cost=", "{:.9f}".format(c), \
...                 "W=", sess.run(W), "b=",
sess.run(b))
...     print("Optimization Finished!")
...     training_cost = sess.run(cost, feed_dict={X:
train_X, Y: train_Y})
...     print("Training cost=", training_cost, "W=",
sess.run(W), "b=", sess.run(b), '\n')
...     # Graphic display
...     plt.plot(train_X, train_Y, 'ro', label='Original
data')
...     plt.plot(train_X, sess.run(W) * train_X +
sess.run(b), label='Fitted line')
...     plt.legend()
```

```
...       plt.show()
...
Epoch: 0050 cost= 20.454841614 W= -0.5383804 b=
-0.39039218
Epoch: 0100 cost= 13.073741913 W= -0.34863567
b= -0.3629717
Epoch: 0150 cost= 8.375241280 W= -0.19726828 b=
-0.34096542
Epoch: 0200 cost= 5.384352684 W= -0.07651979 b=
-0.32327884
Epoch: 0250 cost= 3.480441332 W= 0.019800002 b=
-0.30903846
Epoch: 0300 cost= 2.268455029 W= 0.096629456 b=
-0.29754832
Epoch: 0350 cost= 1.496913791 W= 0.15790887 b=
-0.2882516
Epoch: 0400 cost= 1.005744100 W= 0.2067818 b=
-0.280706
Epoch: 0450 cost= 0.693045378 W= 0.24575654 b=
-0.27455688
Epoch: 0500 cost= 0.493956745 W= 0.2768339 b=
-0.2695219
Epoch: 0550 cost= 0.367188901 W= 0.3016104 b=
-0.26537654
Epoch: 0600 cost= 0.286457956 W= 0.32135987 b=
-0.26194048
Epoch: 0650 cost= 0.235033110 W= 0.33709866 b=
-0.2590707
Epoch: 0700 cost= 0.202264324 W= 0.34963745 b=
```

```
-0.2566525
Epoch: 0750 cost= 0.181371331 W= 0.3596235 b=
-0.2545948
Epoch: 0800 cost= 0.168038592 W= 0.36757308 b=
-0.25282508
Epoch: 0850 cost= 0.159519732 W= 0.37389737 b=
-0.25128493
Epoch: 0900 cost= 0.154065475 W= 0.37892509 b=
-0.2499282
Epoch: 0950 cost= 0.150562286 W= 0.38291833 b=
-0.24871752
Epoch: 1000 cost= 0.148301467 W= 0.38608623 b=
-0.24762374
Optimization Finished!
Training cost= 0.14830147 W= 0.38608623 b=
-0.24762374

[<matplotlib.lines.Line2D object at 0x7f791c0e89e8>]
[<matplotlib.lines.Line2D object at 0x7f7921150860>]
<matplotlib.legend.Legend object at 0x7f791c0fd198>
```

==========

Anlage:

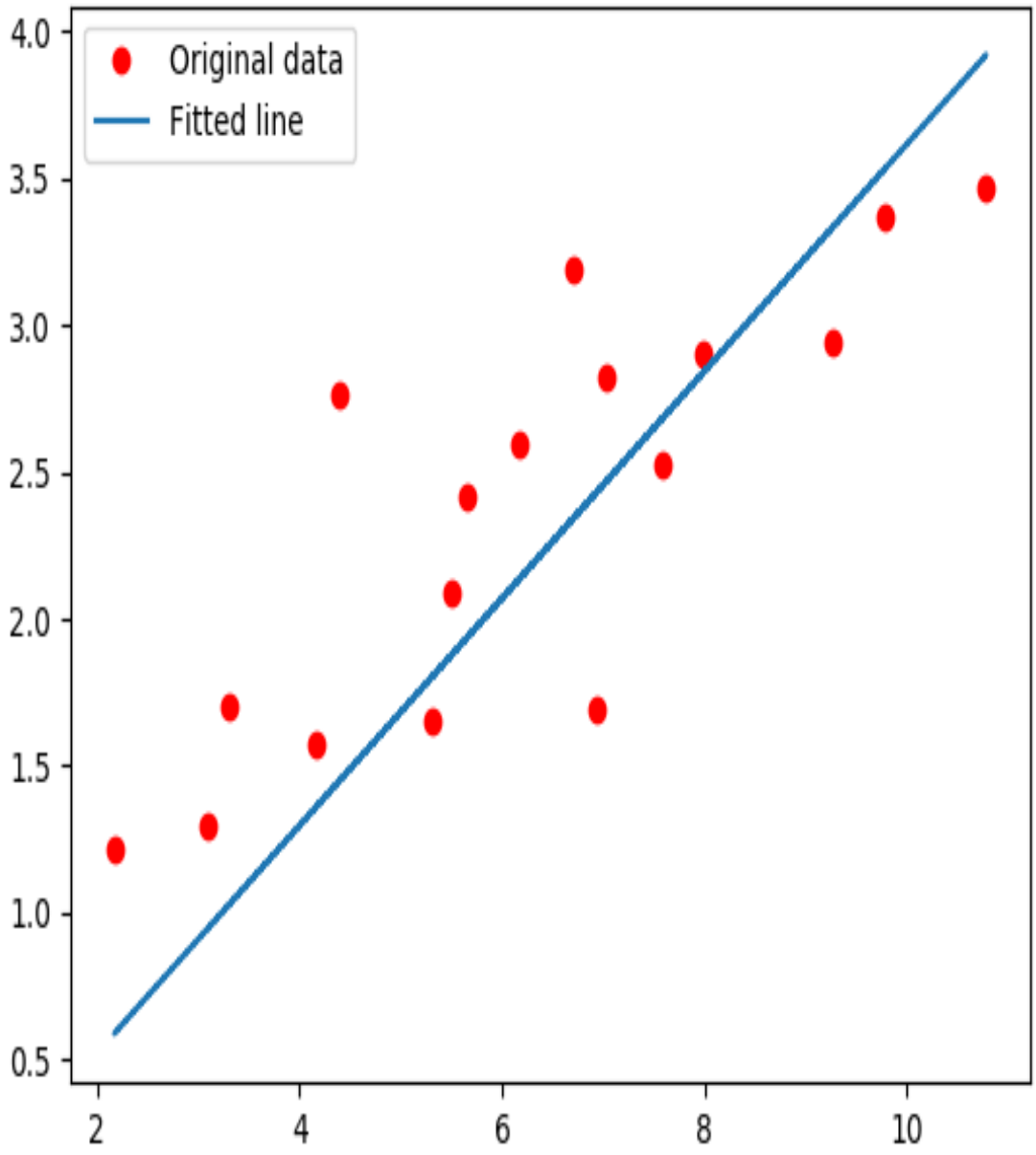
Bilder mit Zielfunktion bei linearer Regression

Download für dieses Dokument:

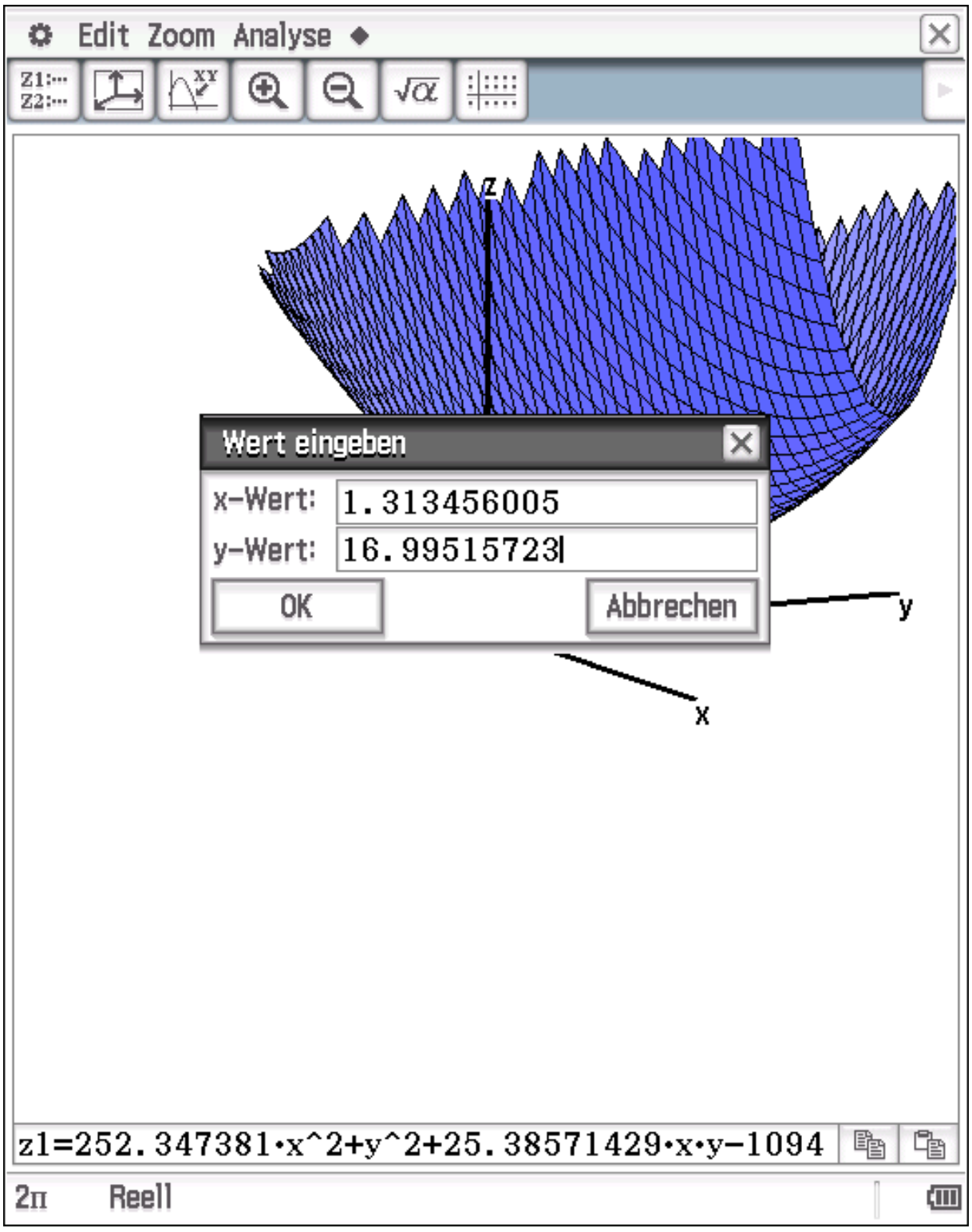
www.informatik.htw-dresden.de/

~paditz/Tensorflow-Ue09.pdf

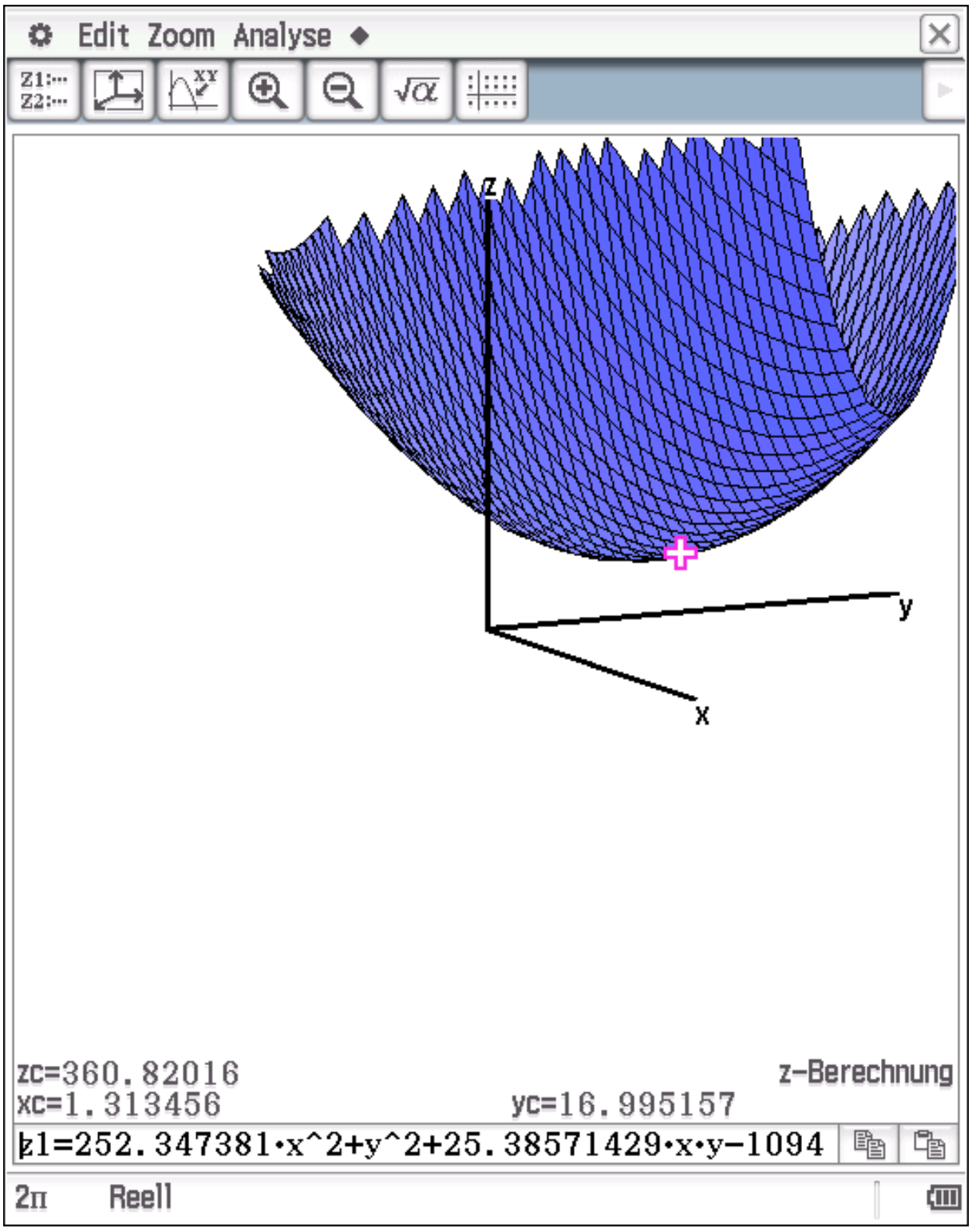
Lineare Regression: (n=17, Trainingsdaten)



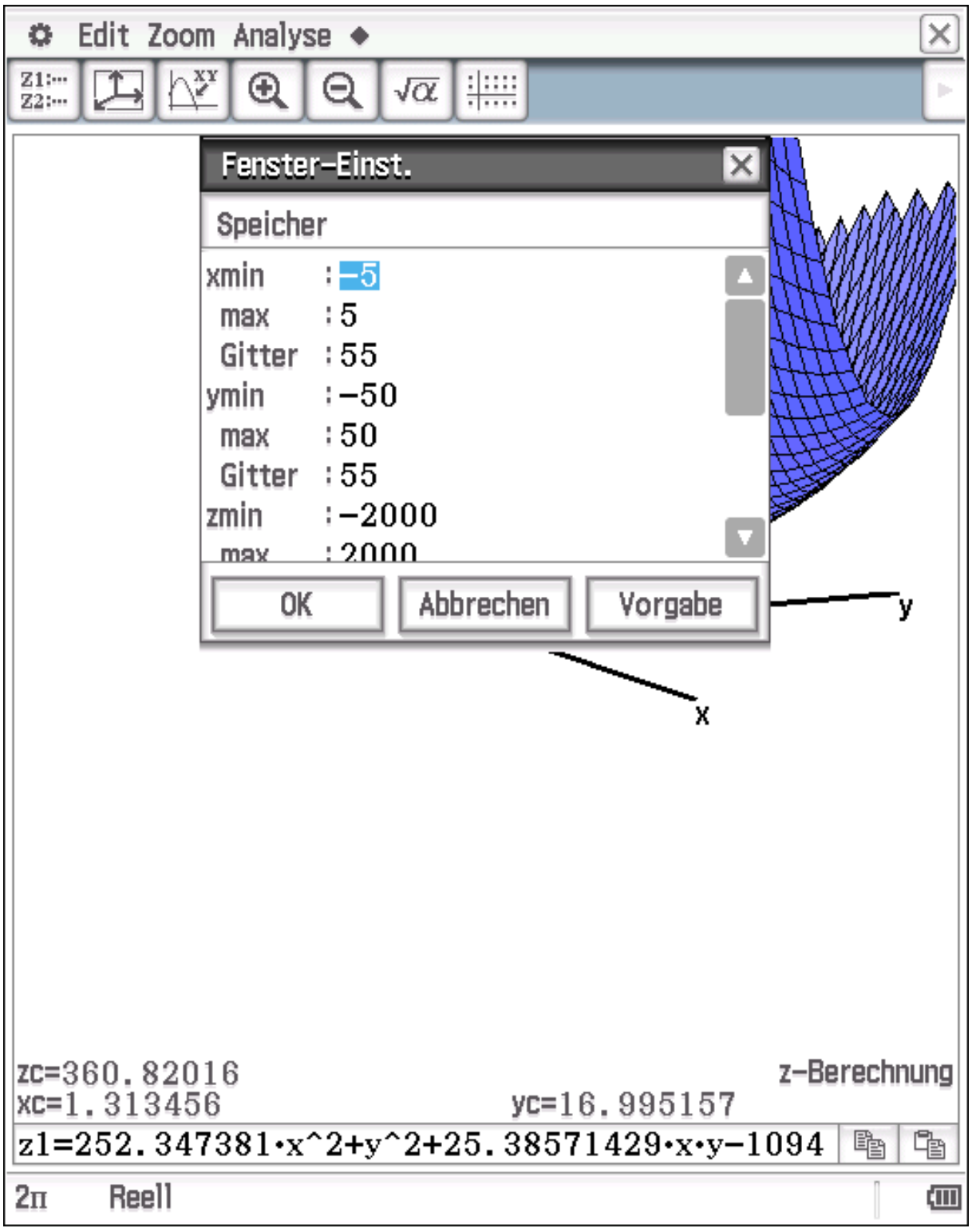
3D-Grafik der Verlustfunktion loss(W,b)



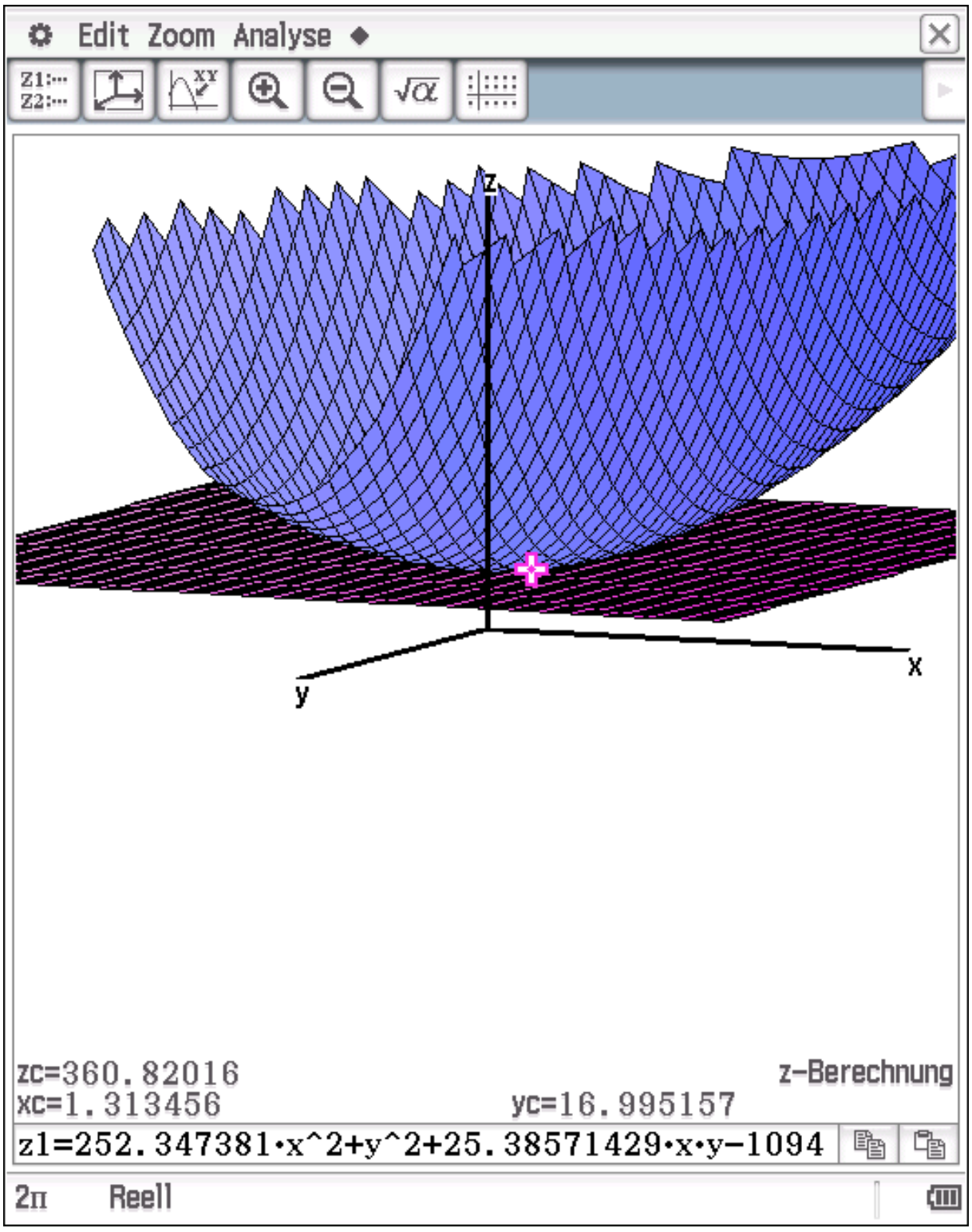
Das globale Min. von loss(W,b) ist erreicht:



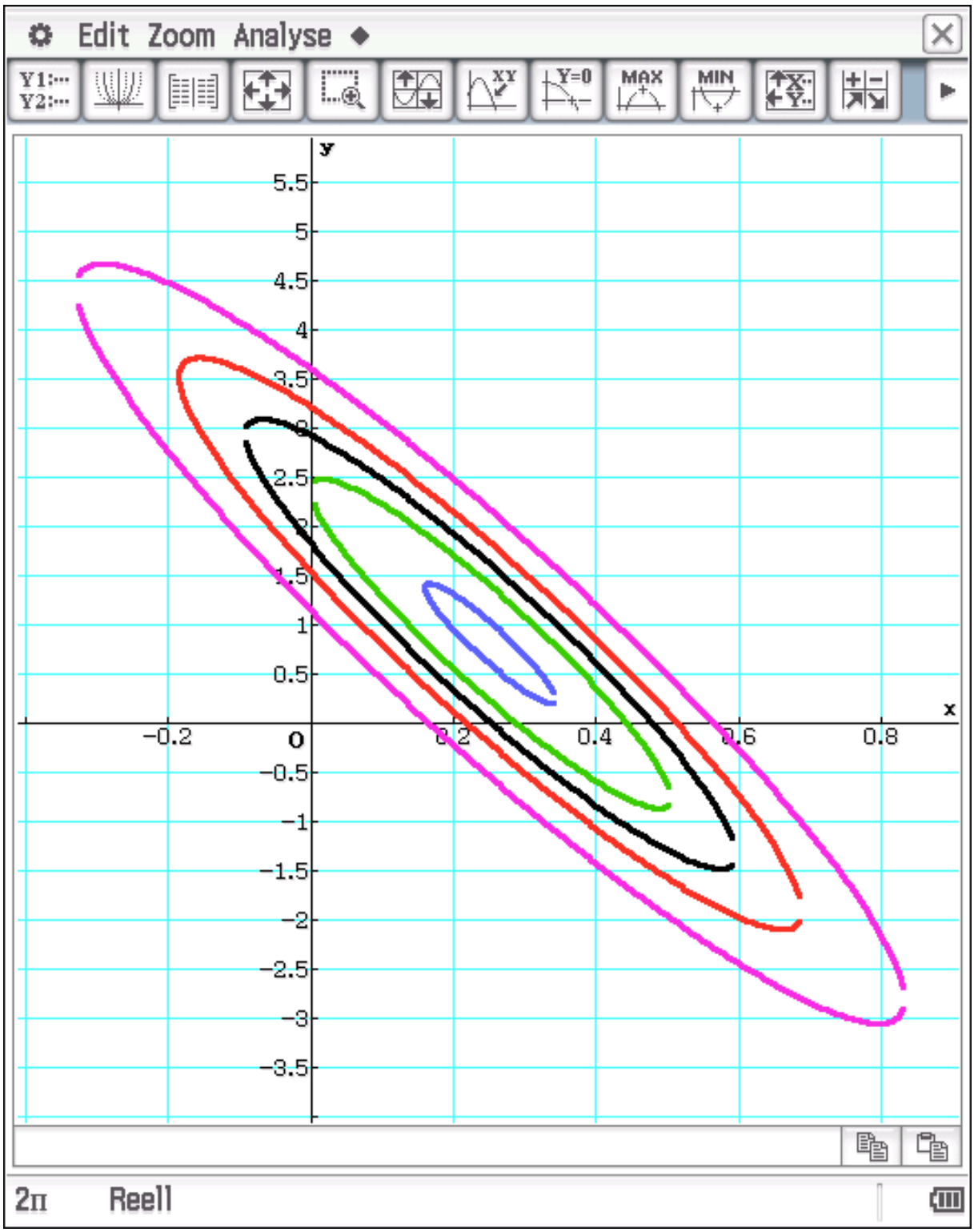
Betrachtungsfenstereinstellung:



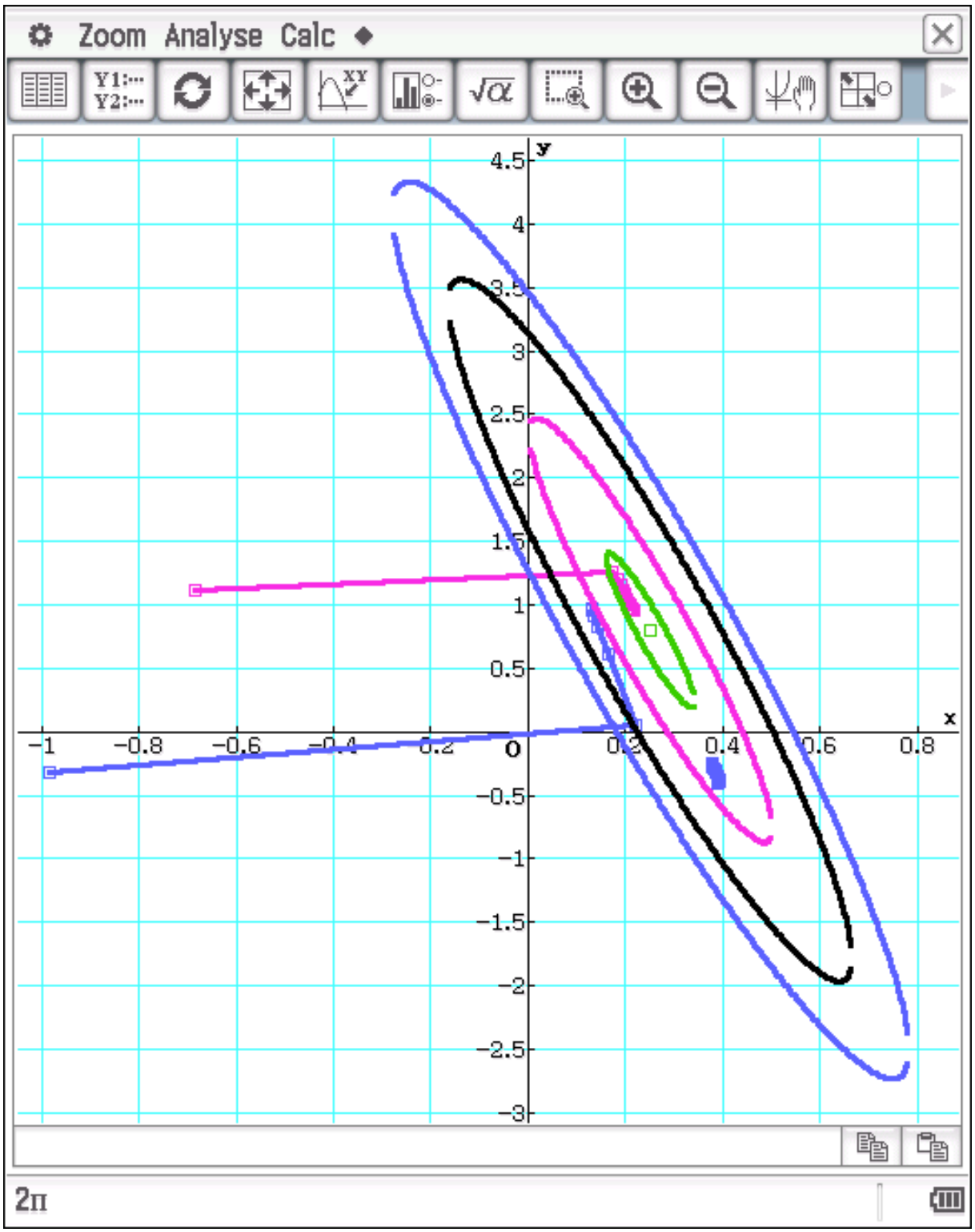
Minimum mit waagerechter Tangentialebene:



Höhenlinien der loss-Funktion:



Iteration mit Tensorflow:



Bei zu großer Schrittweite kann das Verfahren divergieren: