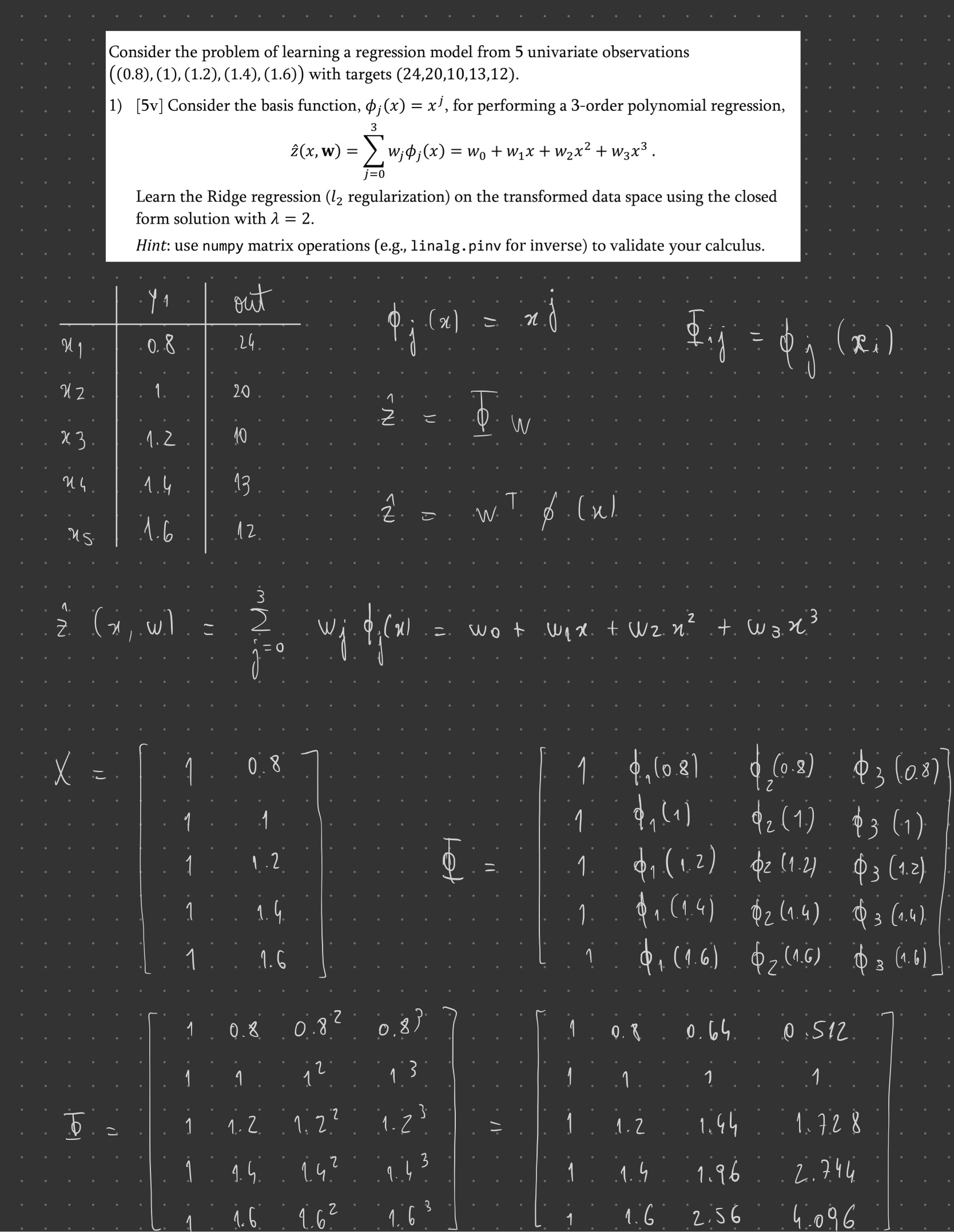
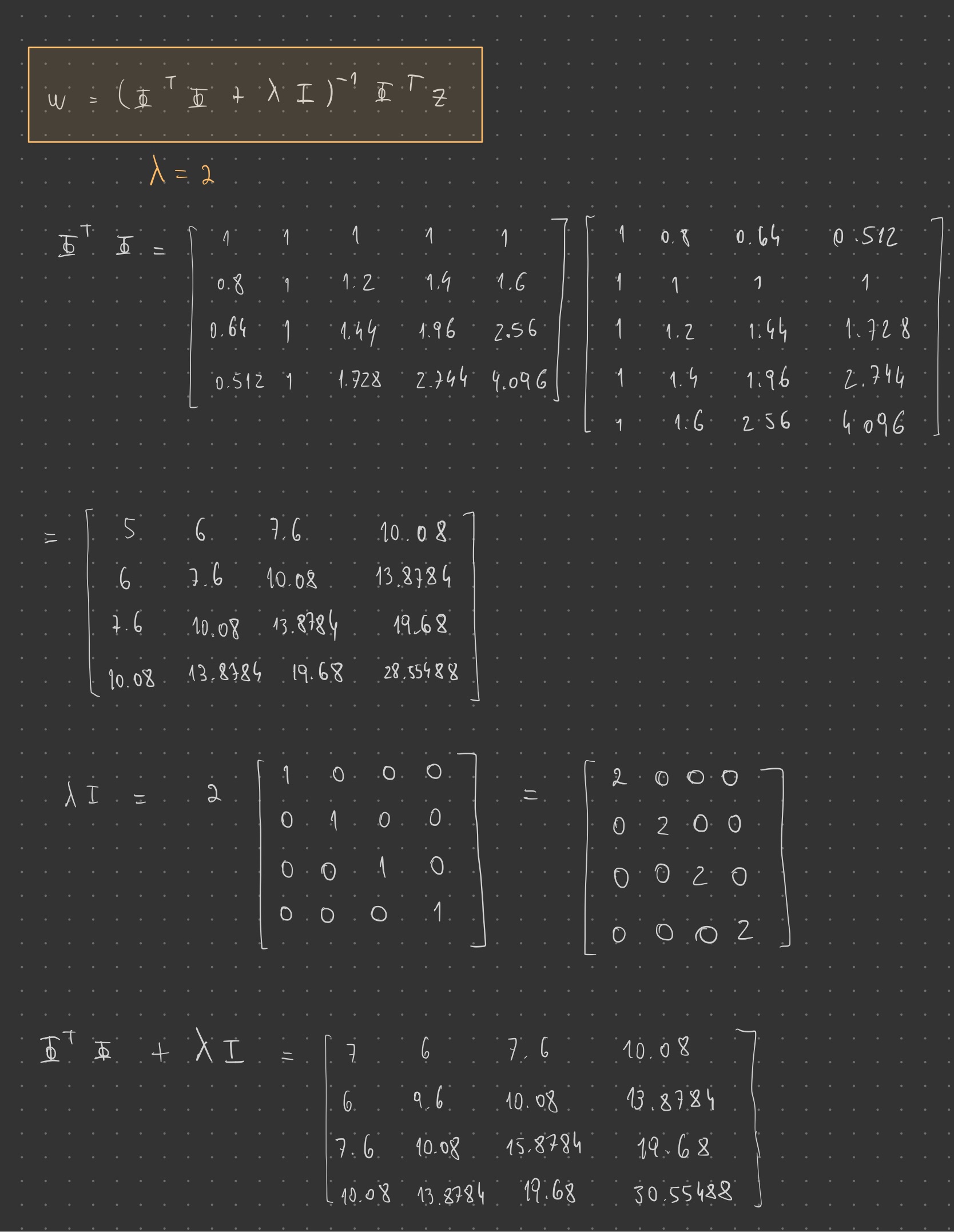
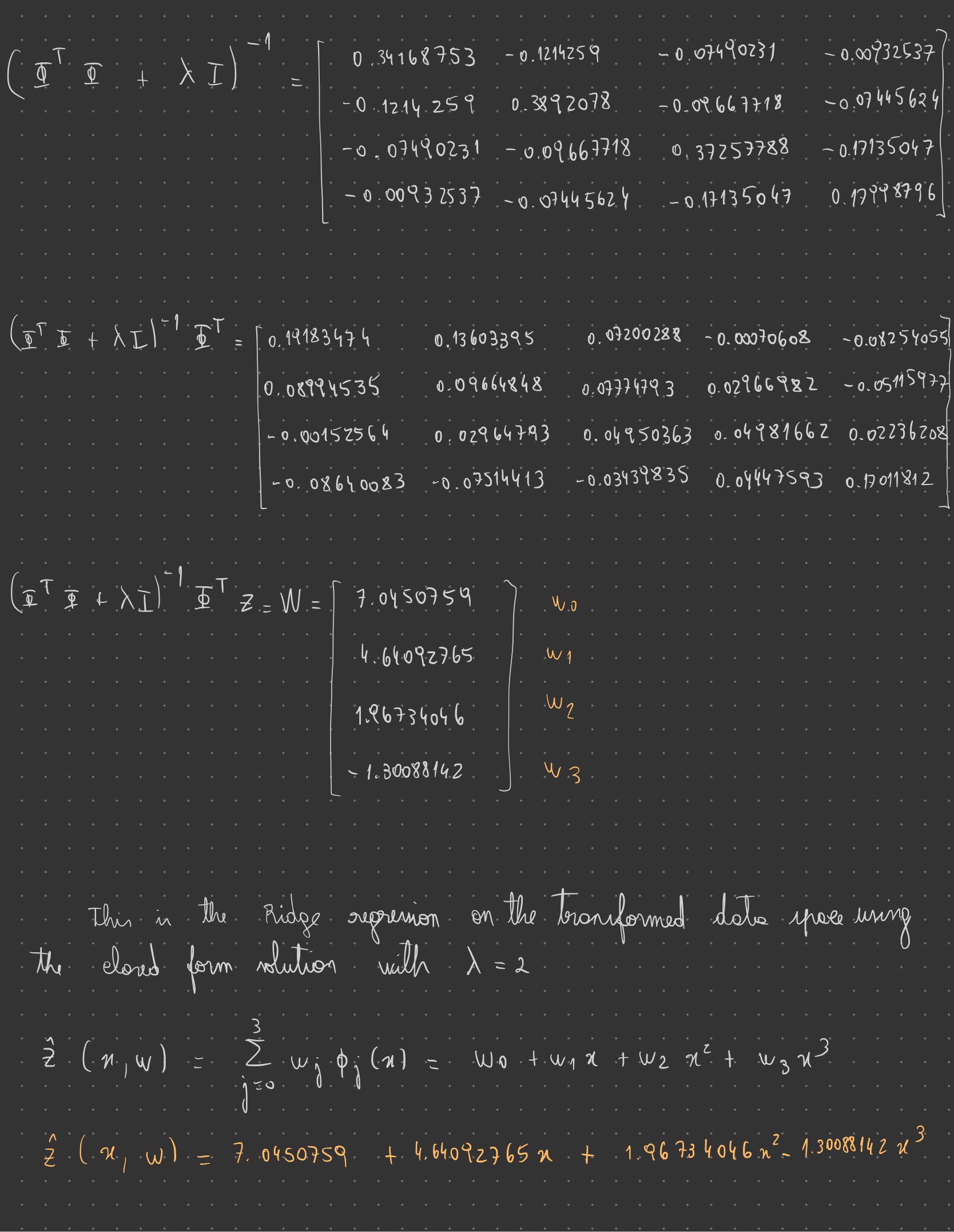
**I. Pen-and-paper**

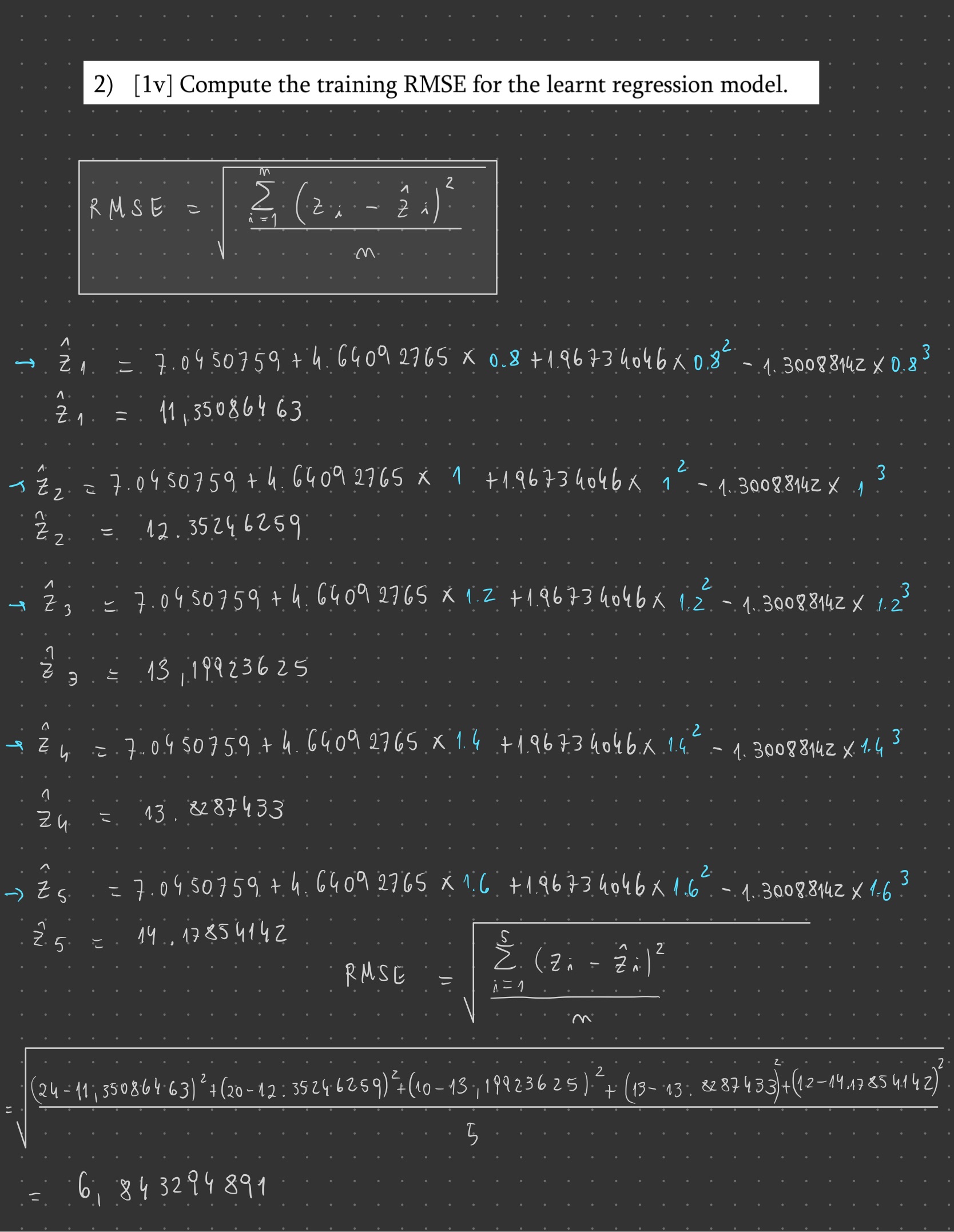




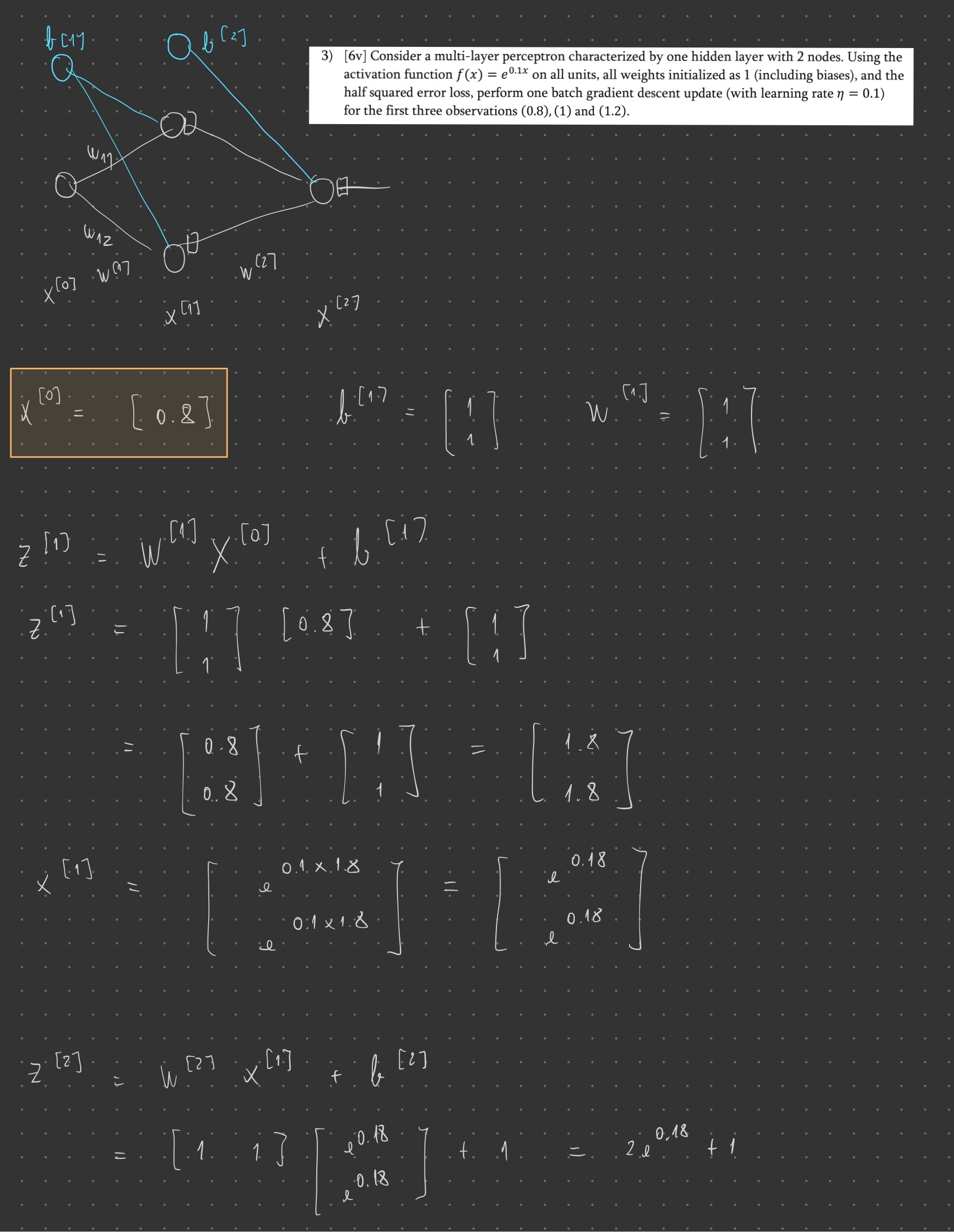


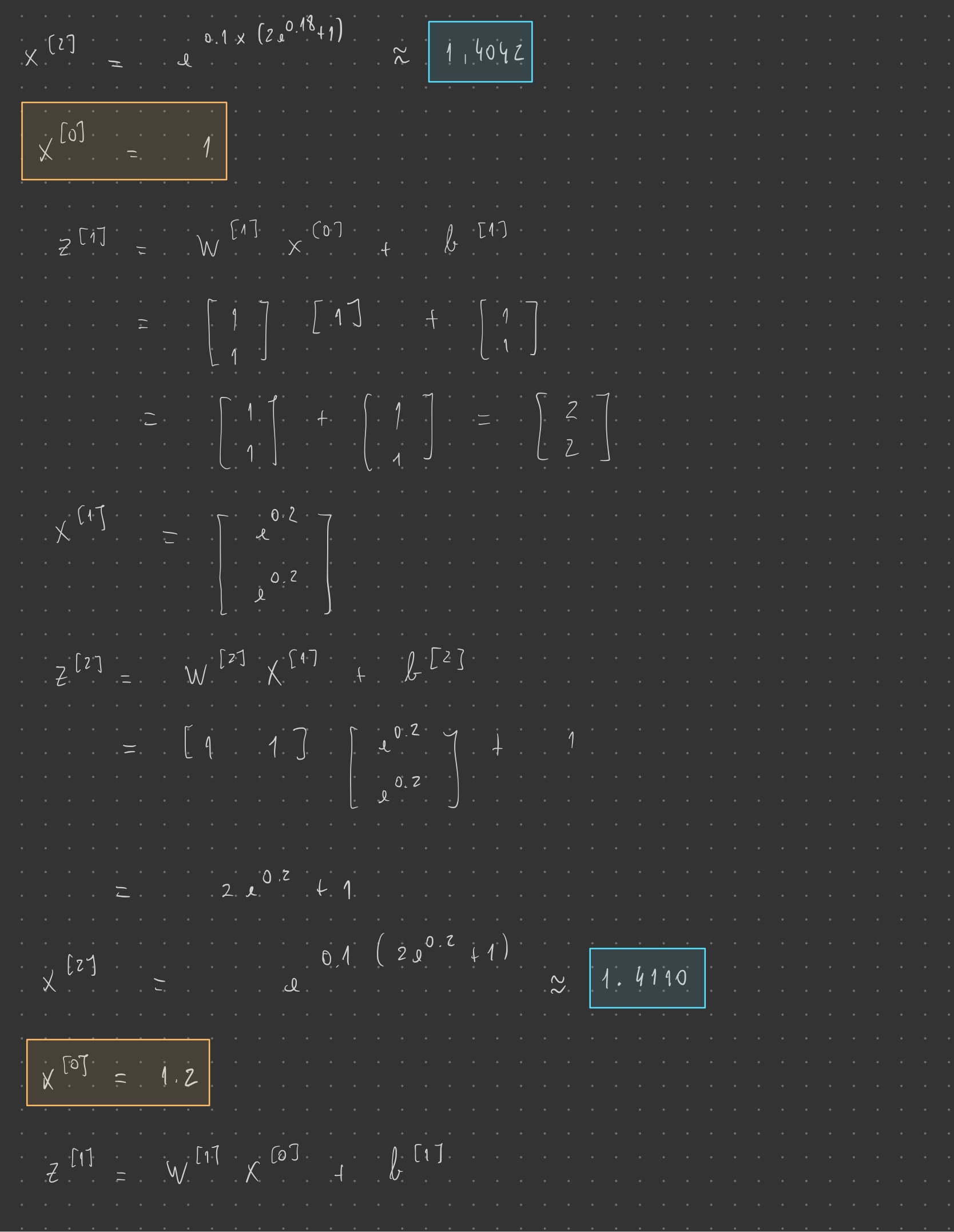


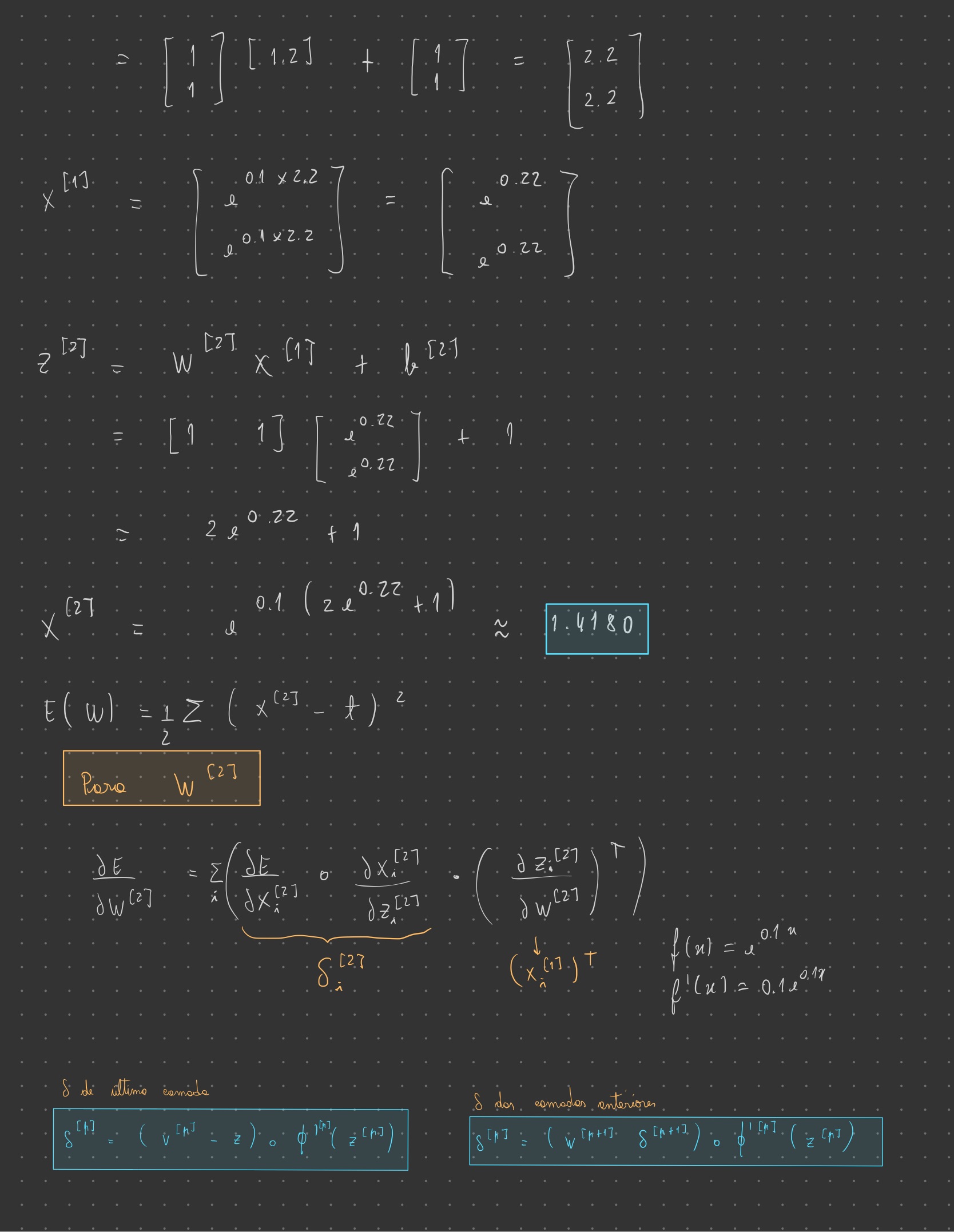
**2)** Answer 2

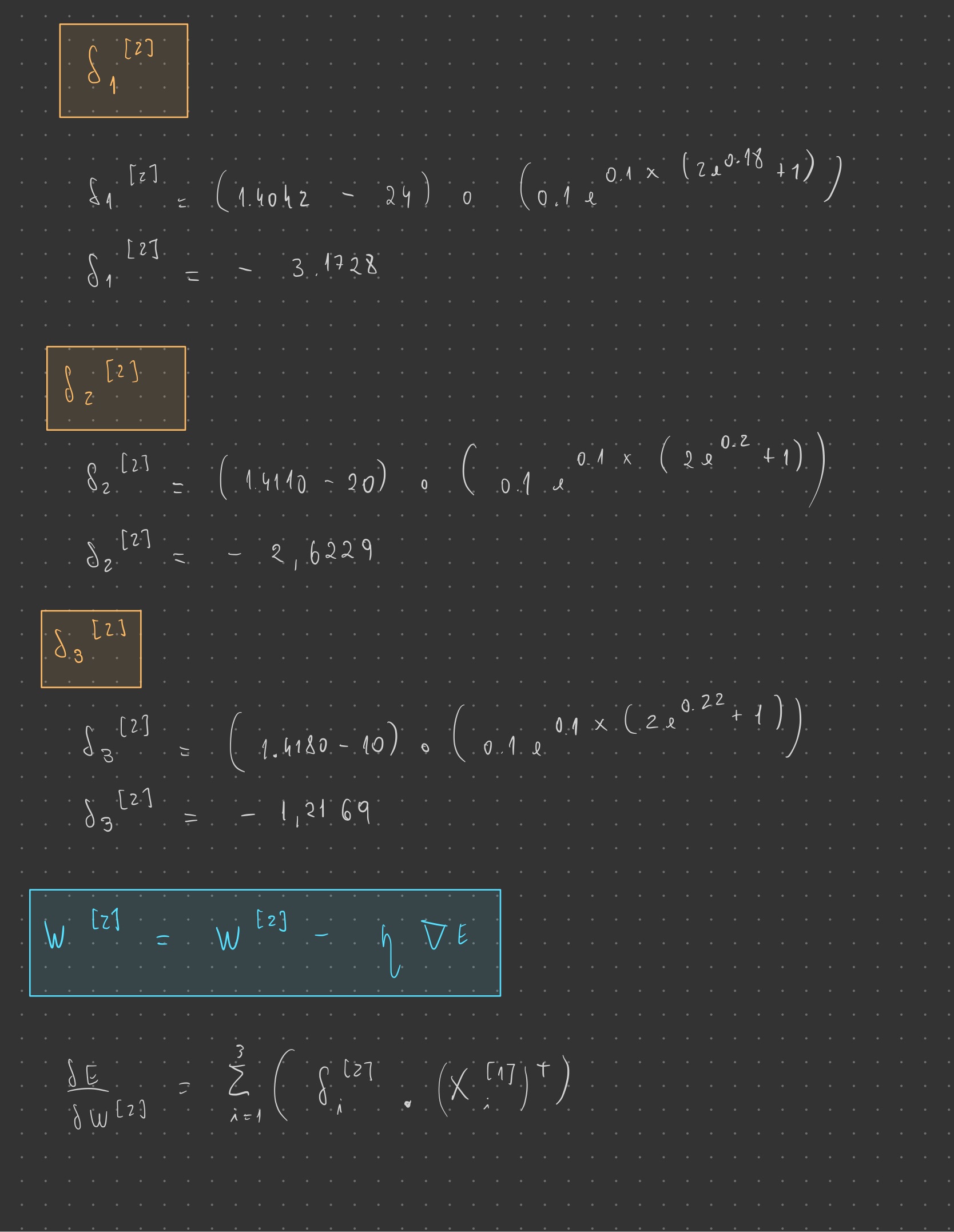


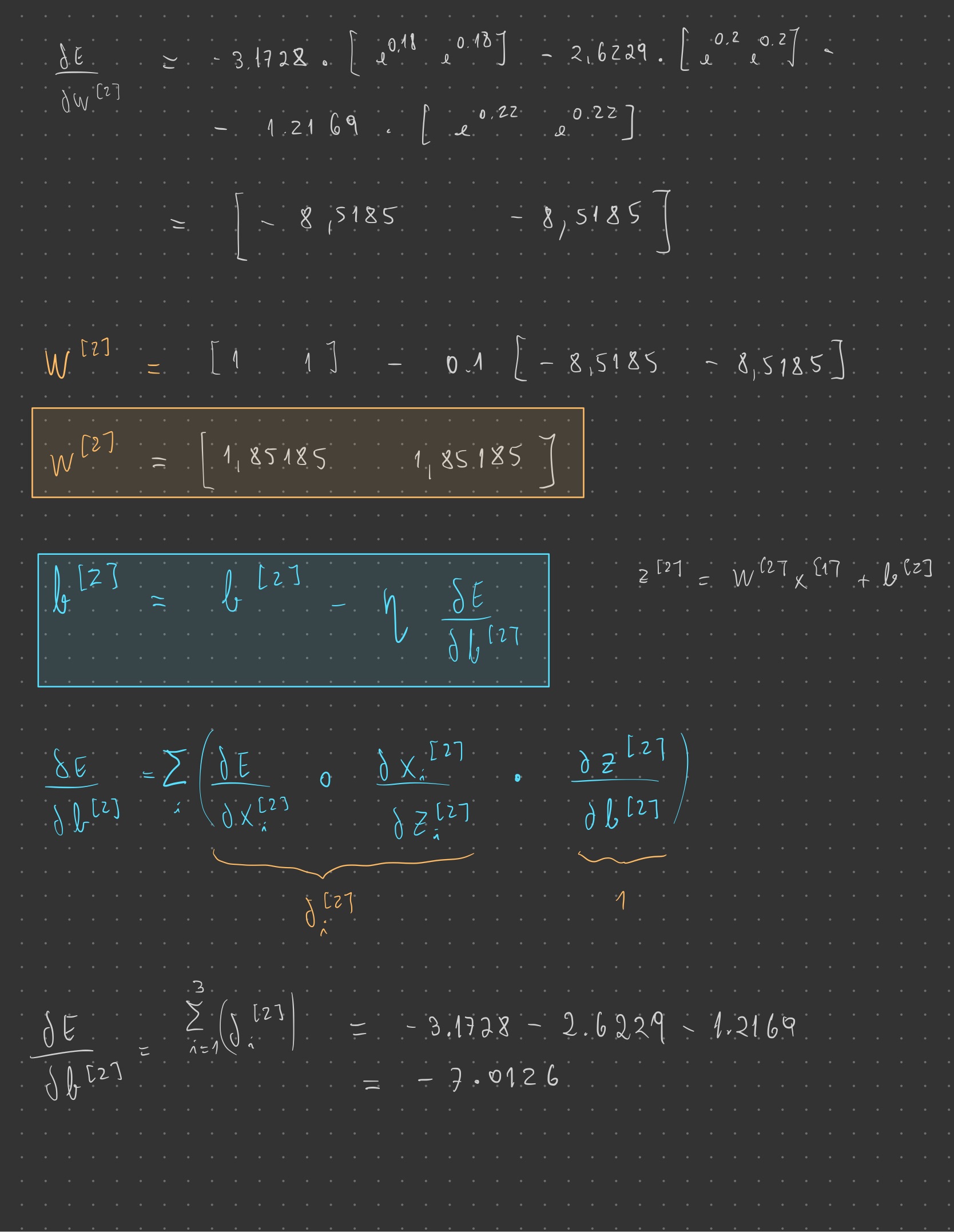
**3)** Answer 3

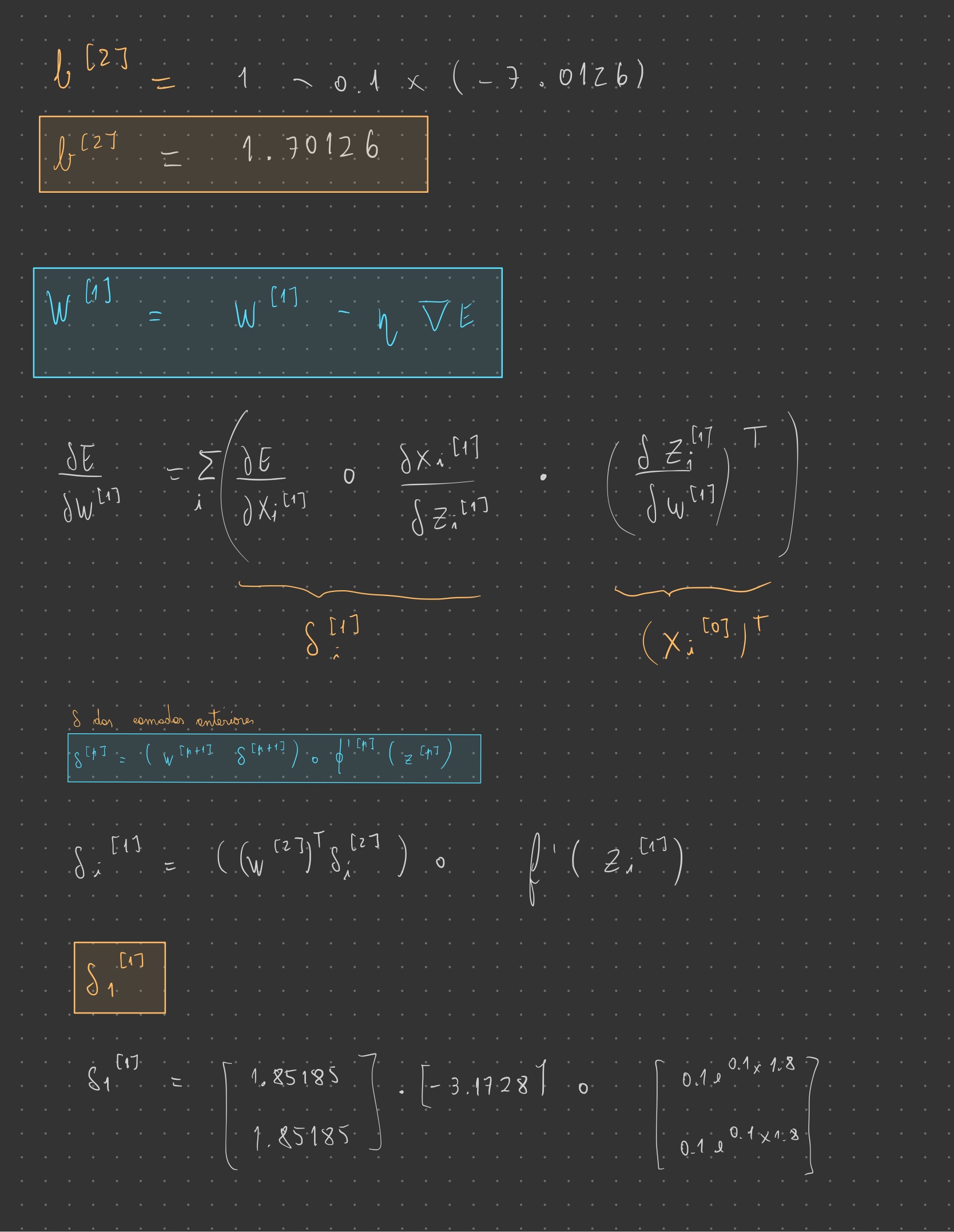
****

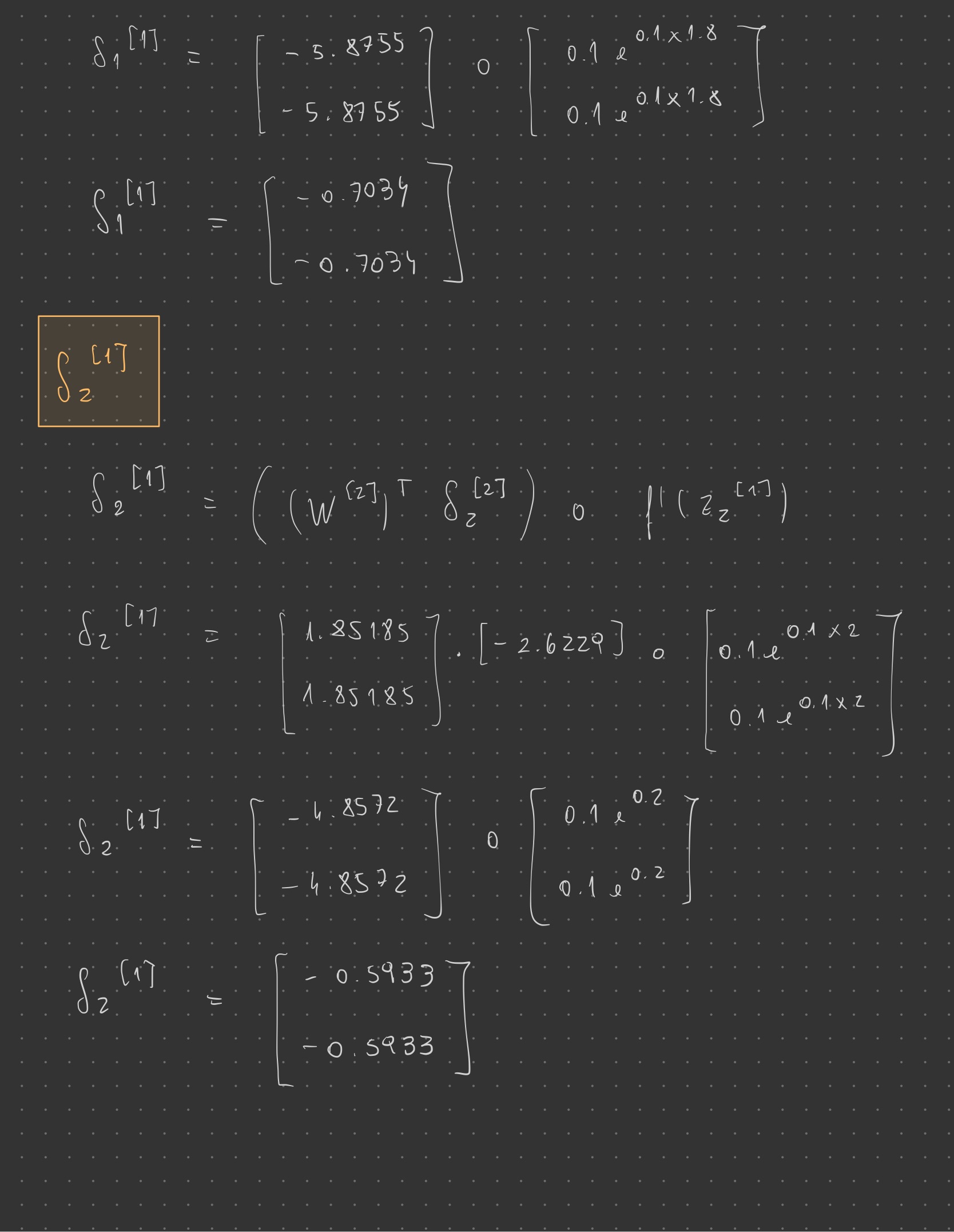
****

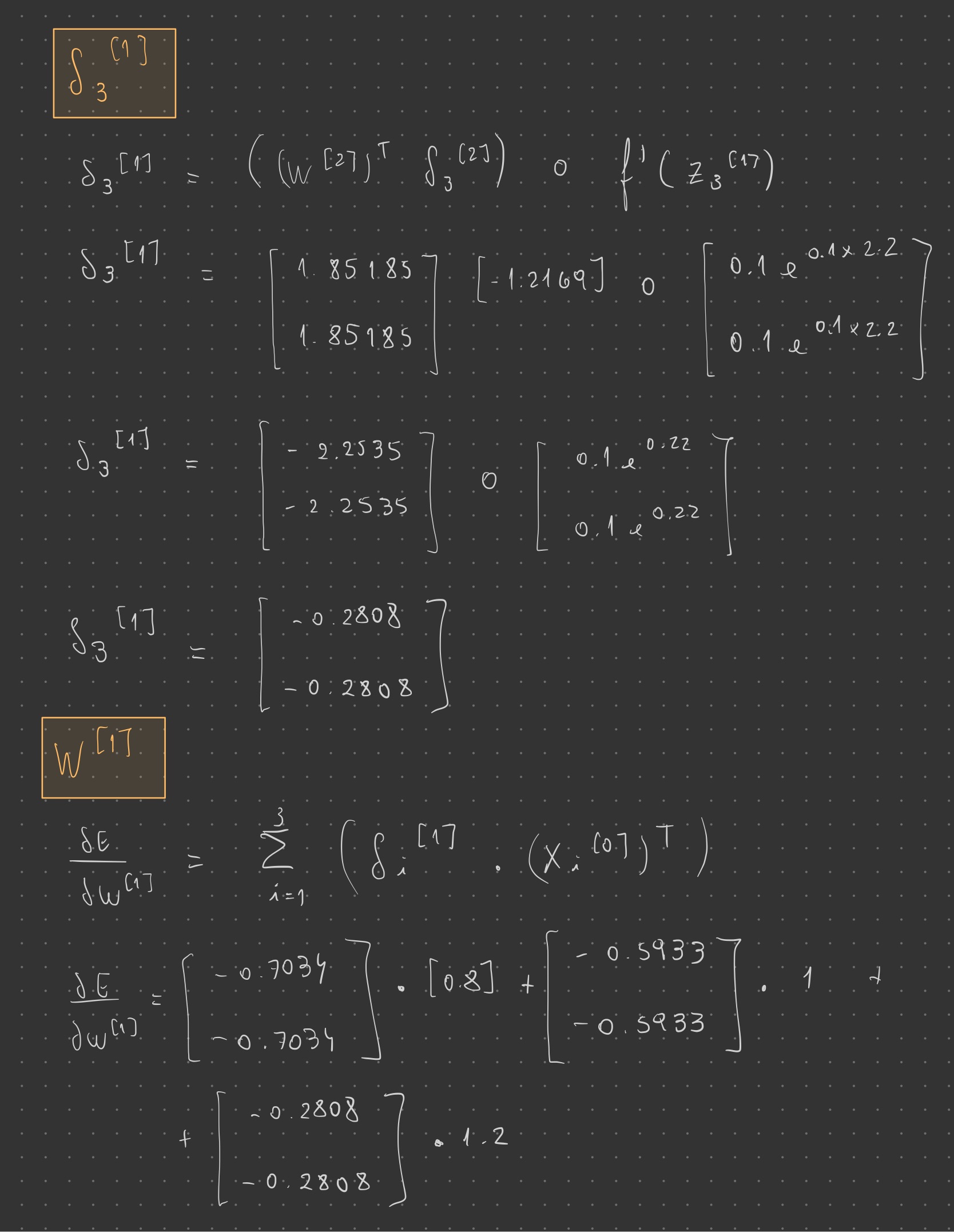
****

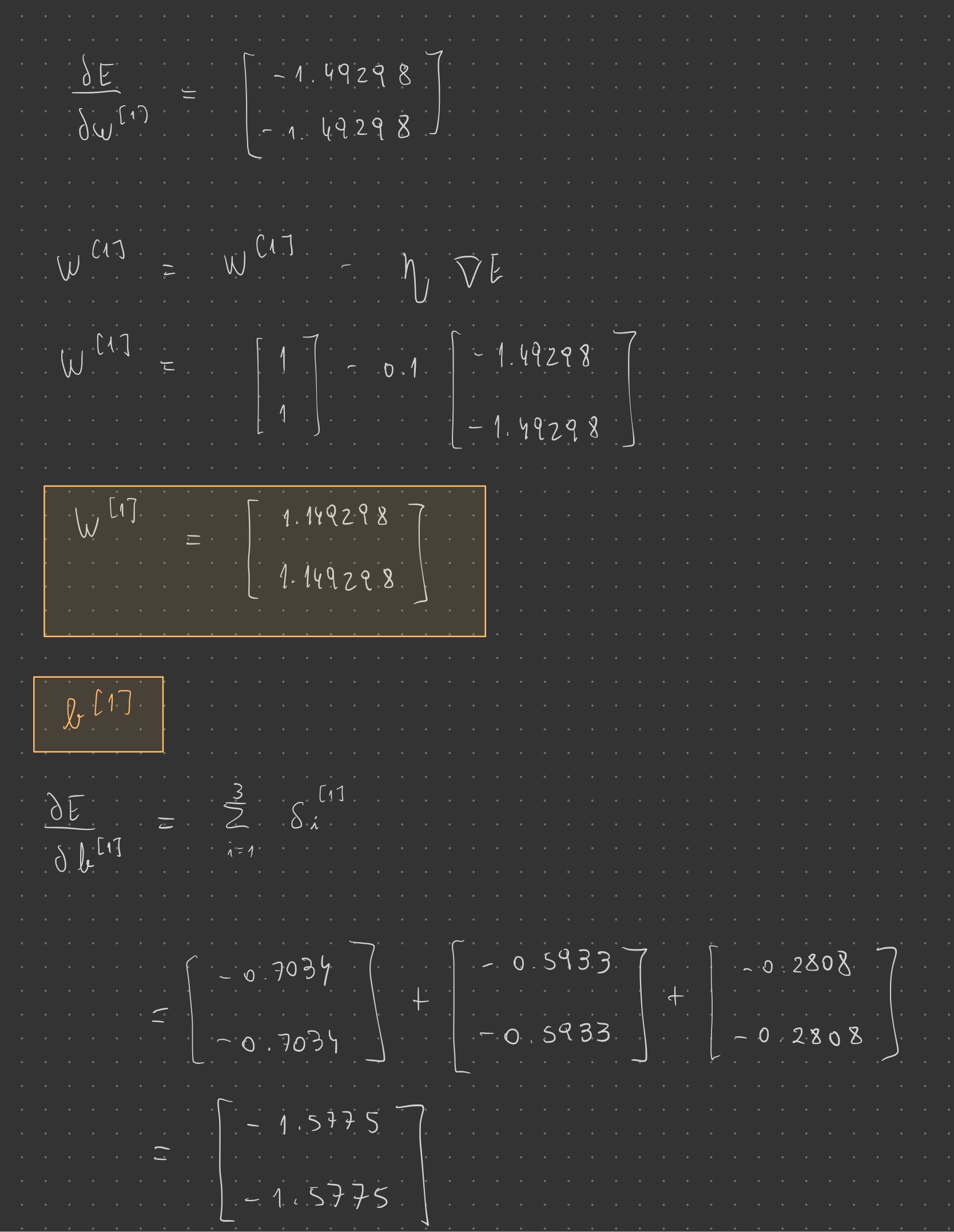
****

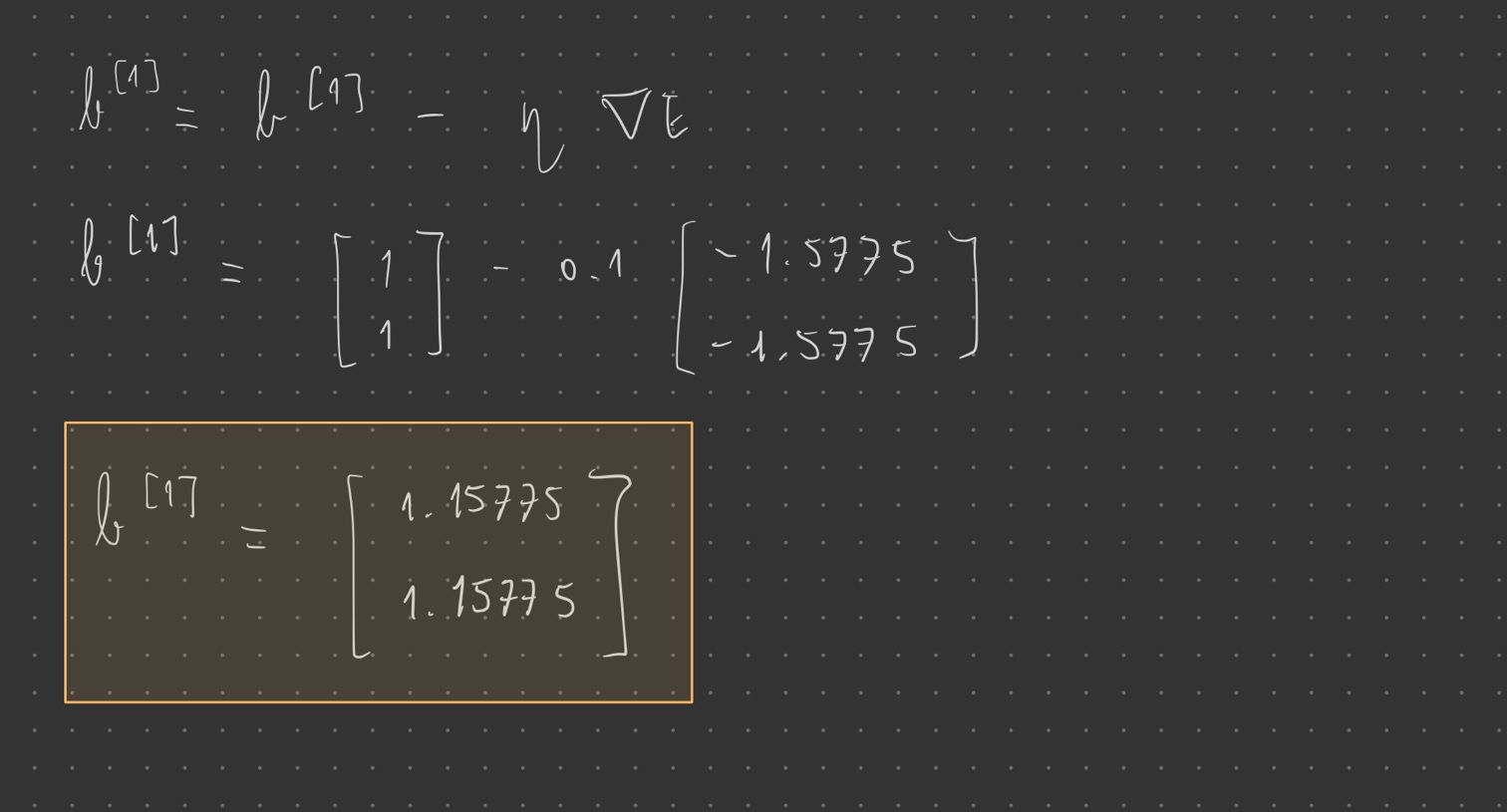
****

****

****

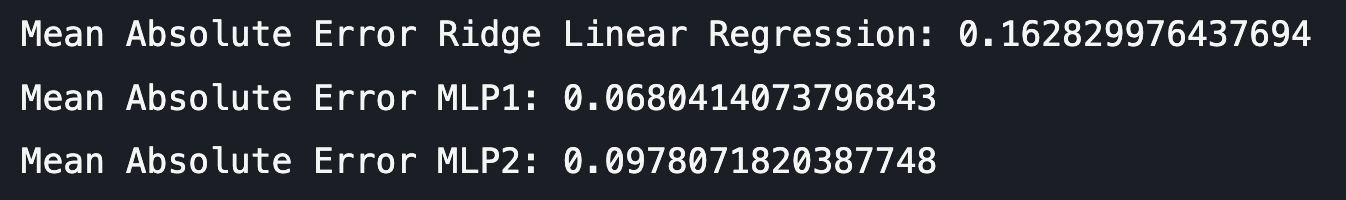
****

****

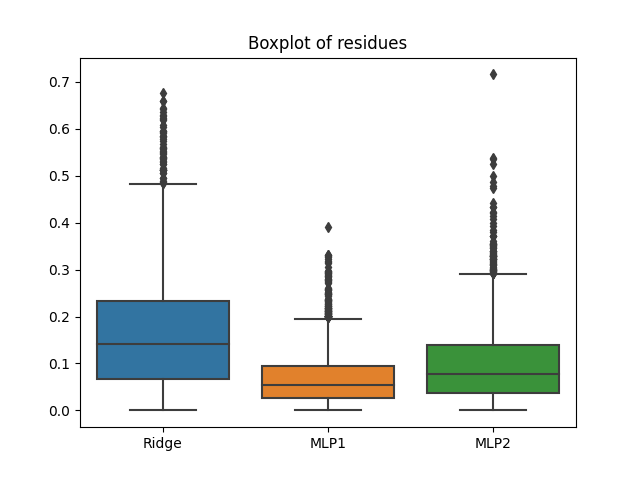
****

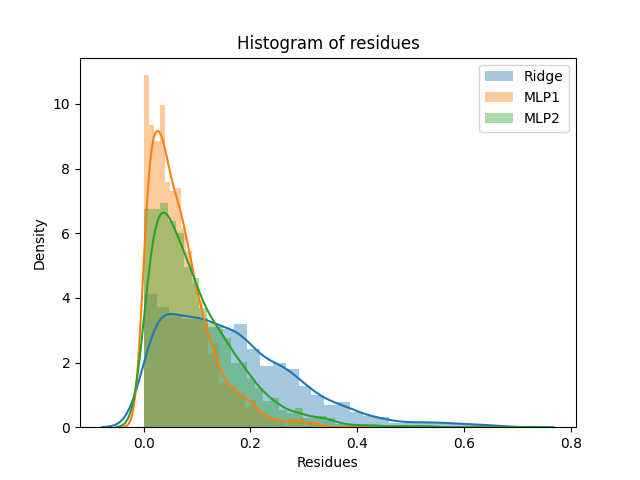
**II. Programming and critical analysis**

**4)** Answer 4

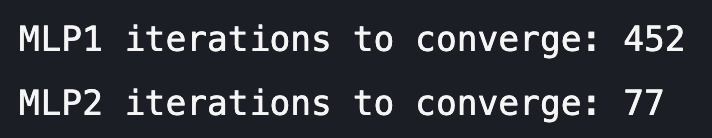


**5)** Answer 5





**6)** Answer 6



**7)** Answer 7

What is motivating the difference between the number of iterations of both MLPs is the early stopping. The fact that the first MLP is parameterized with early stopping helps fighting overfitting. By doing this, we prevent the algorithm from getting too accustomed to the training data and, therefore, it needs more iterations to converge, which makes sense as it is consistently being interrupted by the early stopping. The second MLP converges much faster.

Regarding the observed performance differences between the MLPs, the one with early stopping demonstrates lower average residues and a lower Mean Absolute Error (MAE), which could be due to the fact that, as we fight overfitting, the trained data is better “prepared” when it comes to predicting the outcome of the testing data. On the other hand, the second MLP shows higher average residues and a higher MAE which could be a direct consequence of overfitting.

**III. APPENDIX**

**from** sklearn.linear\_model **import** LinearRegression, Ridge, Lasso

**import** pandas **as** pd

**from** scipy.io.arff **import** loadarff

**from** sklearn.model\_selection **import** train\_test\_split

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** sklearn.neural\_network **import** MLPRegressor

**from** sklearn **import** metrics, datasets

data **=** loadarff('kin8nm.arff')

df **=** pd.DataFrame(data[**0**])

X **=** df.drop('y', **axis=1**)

y **=** df['y']

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, **test\_size** **=** **0.30**, **random\_state** **=** **0**)

*#Ridge Regression*

ridge **=** Ridge(**alpha** **=** **0.1**)

ridge.fit(X\_train, y\_train)

y\_pred\_Ridge **=** ridge.predict(X\_test)

**print**('Mean Absolute Error Ridge Linear Regression:', metrics.mean\_absolute\_error(y\_test, y\_pred\_Ridge))

*#MLP1*

mlp1 **=** MLPRegressor(**hidden\_layer\_sizes** **=** (**10**, **10**), **activation** **=** 'tanh', **max\_iter** **=** **500**, **random\_state** **=** **0**, **early\_stopping** **=** True)

mlp1.fit(X\_train.values, y\_train)

y\_pred\_mlp1 **=** mlp1.predict(X\_test.values)

**print**('Mean Absolute Error MLP1:', metrics.mean\_absolute\_error(y\_test, y\_pred\_mlp1))

*#MLP2*

mlp2 **=** MLPRegressor(**hidden\_layer\_sizes** **=** (**10**, **10**), **activation** **=** 'tanh', **max\_iter=500**, **random\_state** **=** **0**, **early\_stopping** **=** False)

mlp2.fit(X\_train.values, y\_train)

y\_pred\_mlp2 **=** mlp2.predict(X\_test.values)

**print**('Mean Absolute Error MLP2:', metrics.mean\_absolute\_error(y\_test, y\_pred\_mlp2))

*#Boxplots*

ridgeResidues **=** **abs**(y\_test **-** y\_pred\_Ridge)

MLP1Residues **=** **abs**(y\_test **-** y\_pred\_mlp1)

MLP2Residues **=** **abs**(y\_test **-** y\_pred\_mlp2)

residues **=** pd.DataFrame({"Ridge": ridgeResidues, "MLP1": MLP1Residues, "MLP2": MLP2Residues})

sns.boxplot(**data** **=** residues)

plt.title('Boxplot of residues')

plt.savefig('boxplots.png')

plt.show()

*#Histograms*

sns.distplot(residues['Ridge'], **hist** **=** True, **label** **=** 'Ridge')

sns.distplot(residues['MLP1'], **hist** **=** True, **label** **=** 'MLP1')

sns.distplot(residues['MLP2'], **hist** **=** True, **label** **=** 'MLP2')

plt.title('Histogram of residues')

plt.legend()

plt.xlabel('Residues')

plt.savefig('histograms.png')

plt.show()

**print**('MLP1 iterations to converge:', mlp1.n\_iter\_)

**print**('MLP2 iterations to converge:', mlp2.n\_iter\_)

**if** mlp1.n\_iter\_ **<** mlp1.max\_iter:

**print**("MLP1 converged")

**else**:

**print**("MLP1 did not converge")

**if** mlp2.n\_iter\_ **<** mlp2.max\_iter:

**print**("MLP2 converged")

**else**:

**print**("MLP2 did not converge")

**END**