Open in Colab

Welcome to WOA7015 Advance Machine Learning Lab - Week 3

This code is generated for the purpose of WOA7015 module. The code is available in github https://github.com/shiernee/Advanced_ML

The effect of imbalanced data on AUROC

The following code evaluates the effect of imbalanced data on the AUROC of TPR-FPR curve.

```
# roc curve and auc on an imbalanced dataset
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from imblearn.under_sampling import RandomUnderSampler
```

```
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1000)

print(X)
print('-----------')
print(y)
```

```
[[-0.32584935  0.21897754  0.62061895 ...  2.84071377 -0.02582733
  -0.40885762]
 [-1.12624124 -0.86026727 -0.89264356 ... -0.92962064  0.59483549
   1.24052468]
 [-0.48993428 -0.7453348  -1.43801838 ... -1.67525801 -0.09994425
  -0.46569289]
 ...
 [ 0.47406074 -1.9209351   0.41681779 ...  1.04574815  1.092832
  -0.01541749]
 [-0.62731673 -0.94336697 -1.50694171 ... -0.85092941  0.99046917
   2.19583454]
 [ 0.88990126  0.81857103 -2.12551556 ...  1.00271323 -0.88101446
```

```
  -0.81149645]]
-----------
[1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0
 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0
 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1
 0 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1
 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0
 1 0 1 1 0 1 1 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 0
 1 1 1 1 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 0 1 0 1
 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0
 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0
 1 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0
 1 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 1 1 1 0
 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0
 0 1 0 1 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0
 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0
 1 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 1
 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
 0 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 0 0 0
 0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 1 0
 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 0 0 0 1
 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0
 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 1
 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1
 1 1 1 1 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 0
 0 0 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1
 1 1 1 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 0
 1 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0
 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0
 0]
```

```
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=1000)

print('trainy - class0: ', len(trainy)-trainy.sum())
print('trainy - class1: ', trainy.sum())
print('----------------------')
print('testy - class0: ', len(testy)-testy.sum())
print('testy - class1: ', testy.sum())
print('============================')

# make testing dataset balance
undersample = RandomUnderSampler(sampling_strategy='majority')
testX, testy = undersample.fit_resample(testX, testy)

print('Balanced Testing date')
print('testy - class0: ', len(testy)-testy.sum())
print('testy - class1: ', testy.sum())
```

```
trainy - class0:  253
trainy - class1:  247
----------------------
testy - class0:  249
```

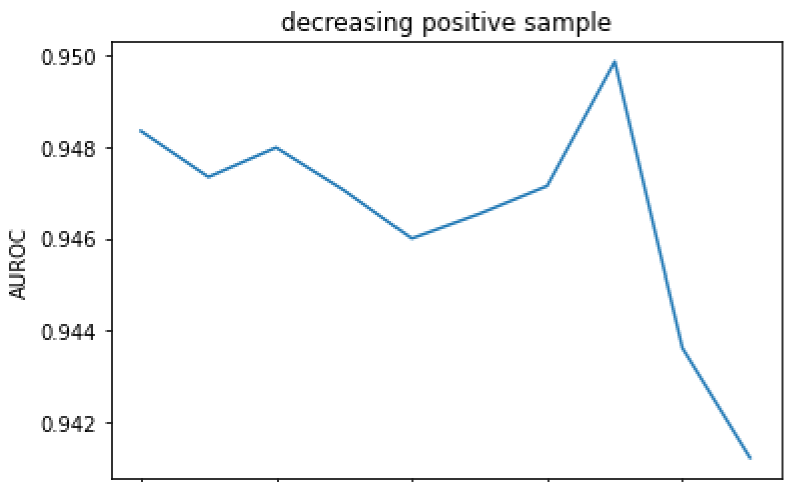
```
testy - class1:  251
============================
Balanced Testing date
testy - class0:  249
testy - class1:  249
```

```
# fit a model with training data
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
```

```
LogisticRegression()
```

```
# repeat with different skewness
roc_list = []
lr_acc = []
k=1
for i in range(0, 10):
    pos_ind = np.where(testy==1)[0]
    n = int(i/10 * len(pos_ind))
    tmp_testX, tmp_testy = np.copy(testX), np.copy(testy)
    tmp_testX = np.delete(tmp_testX, pos_ind[:n], axis=0)
    tmp_testy = np.delete(tmp_testy, pos_ind[:n], axis=0)
    print('nth %d:positive: %d negative: %d'
            % (i, tmp_testy.sum(), tmp_testy.shape[0] - tmp_testy.sum()))
    print('---------------------------------------------')

    # predict probabilities
    lr_probs = model.predict_proba(tmp_testX)
    # keep probabilities for the positive outcome only
    lr_probs = lr_probs[:, 1]
    # calculate scores
    lr_auc = roc_auc_score(tmp_testy, lr_probs)

    # summarize scores
    # print('iteration %d: Logistic: ROC AUC=%.3f' % (k, lr_auc))
    k += 1
    # calculate roc curves
    lr_fpr, lr_tpr, _ = roc_curve(tmp_testy, lr_probs)
    roc_list.append(lr_auc)

plt.plot(np.arange(0, len(roc_list)), roc_list)
plt.xlabel('skewness ratio')
plt.ylabel('AUROC')
plt.title('decreasing positive sample')
```

```
nth 0:positive: 249 negative: 249
---------------------------------------------
nth 1:positive: 225 negative: 249
---------------------------------------------
nth 2:positive: 200 negative: 249
---------------------------------------------
nth 3:positive: 175 negative: 249
---------------------------------------------
nth 4:positive: 150 negative: 249
---------------------------------------------
nth 5:positive: 125 negative: 249
---------------------------------------------
nth 6:positive: 100 negative: 249
---------------------------------------------
nth 7:positive: 75 negative: 249
---------------------------------------------
nth 8:positive: 50 negative: 249
---------------------------------------------
nth 9:positive: 25 negative: 249
---------------------------------------------
Text(0.5, 1.0, 'decreasing positive sample')
```



Exercise 1 (2%):

Does the AUROC (TPR vs FPR) affected by imbalanced class?

```
# Your answer here

### NO, the AUROC is not affected by imbalanced class. The AUROC is
### not reflected by data imbalance. As the AUROC is a measure of
### recall, the imbalanced class doesn't matter. Also, the changes
### in the class distribution has no influence on the AUROC.
```

The effect of imbalanced data on AUROC of PR curve and F1 score

The following code evaluates the effect of imbalanced data on the AUROC of Precision-Recall and F1 value.

```
# roc curve and auc on an imbalanced dataset
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import auc, f1_score
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt
```

```
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1000)

print(X)
print('-----------')
print(y)
```

```
[[-0.32584935  0.21897754  0.62061895 ...  2.84071377 -0.02582733
  -0.40885762]
 [-1.12624124 -0.86026727 -0.89264356 ... -0.92962064  0.59483549
   1.24052468]
 [-0.48993428 -0.7453348  -1.43801838 ... -1.67525801 -0.09994425
  -0.46569289]
 ...
 [ 0.47406074 -1.9209351   0.41681779 ...  1.04574815  1.092832
  -0.01541749]
 [-0.62731673 -0.94336697 -1.50694171 ... -0.85092941  0.99046917
   2.19583454]
 [ 0.88990126  0.81857103 -2.12551556 ...  1.00271323 -0.88101446
  -0.81149645]]
-----------
[1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0
 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 1 1 1 0 0 0 0
 0 0 0 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 0 0 1
 0 1 1 0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 1 0 0 0 0 1 0 1
 0 0 1 1 0 0 1 1 1 0 1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0
 1 0 1 1 0 1 1 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 0
 1 1 1 1 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0 1 0 0 0 1 0 1
 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 1 1 0 1 1 1 1 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0
 1 1 0 0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 0
 1 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0
 1 1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 1 0 1 0 1 1 1 0
 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1 0 0 0 1 0 1 1 0 0
 0 1 0 1 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0
 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0
 1 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 1
 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
```

```
 0 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 0 0 0
 0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 1 1 0 0 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 1 0
 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 0 0 0 1
 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0
 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 1
 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 1
 1 1 1 1 0 0 0 1 0 1 1 0 1 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 0
 0 0 0 1 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1
 1 1 1 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 1 1 0 0 1 1 0
 1 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 0
 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1 0
 0]
```

```
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=1000)

print('trainy - class0: ', len(trainy)-trainy.sum())
print('trainy - class1: ', trainy.sum())
print('----------------------')
print('testy - class0: ', len(testy)-testy.sum())
print('testy - class1: ', testy.sum())
print('===========================')

# make testing dataset balance
undersample = RandomUnderSampler(sampling_strategy='majority')
testX, testy = undersample.fit_resample(testX, testy)

print('Balanced Testing date')
print('testy - class0: ', len(testy)-testy.sum())
print('testy - class1: ', testy.sum())
```

```
trainy - class0:  253
trainy - class1:  247
----------------------
testy - class0:  249
testy - class1:  251
===========================
Balanced Testing date
testy - class0:  249
testy - class1:  249
```

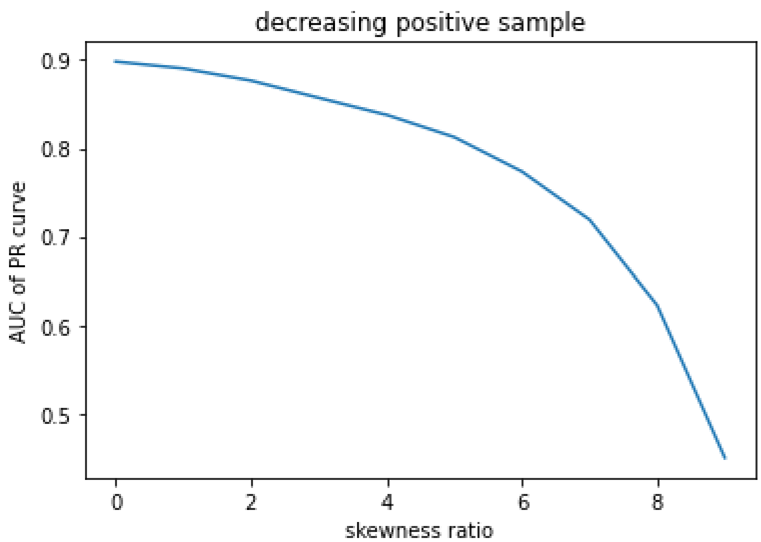
```
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
```

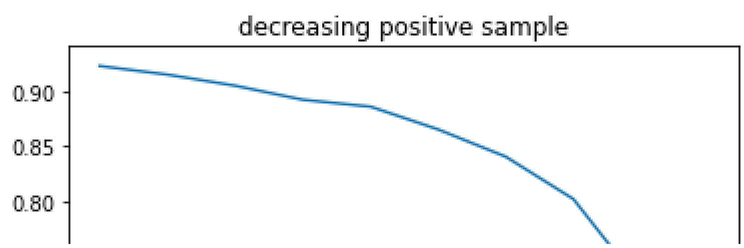
```
LogisticRegression()
```

```
# repeat with different skewness
roc_list = []
f1_list = []
```

```
k=1
for i in range(0, 10):
    pos_ind = np.where(testy==1)[0]
    n = int(i/10 * len(pos_ind))
    tmp_testX, tmp_testy = np.copy(testX), np.copy(testy)
    tmp_testX = np.delete(tmp_testX, pos_ind[:n], axis=0)
    tmp_testy = np.delete(tmp_testy, pos_ind[:n], axis=0)
    print('nth %d:positive: %d negative: %d'
            % (i, tmp_testy.sum(), tmp_testy.shape[0] - tmp_testy.sum()))
    print('---------------------------------------------')


    # predict probabilities
    lr_probs = model.predict_proba(tmp_testX)
    # keep probabilities for the positive outcome only
    lr_probs = lr_probs[:, 1]
    # predict class values
    yhat = model.predict(tmp_testX)
    # calculate precision and recall for each threshold
    lr_precision, lr_recall, _ = precision_recall_curve(tmp_testy, lr_probs)
    # calculate scores
    lr_f1, lr_auc = f1_score(tmp_testy, yhat), auc(lr_recall, lr_precision)
    # summarize scores
    # print('iteration%d Logistic: f1=%.3f auc=%.3f' % (k, lr_f1, lr_auc))
    k += 1
    roc_list.append(lr_auc)
    f1_list.append(lr_f1)

plt.plot(np.arange(0, len(roc_list)), roc_list)
plt.xlabel('skewness ratio')
plt.ylabel('AUC of PR curve')
plt.title('decreasing positive sample')

plt.figure()
plt.plot(np.arange(0, len(roc_list)), f1_list)
plt.xlabel('skewness ratio')
plt.ylabel('F1')
plt.title('decreasing positive sample')
```

```
nth 0:positive: 249 negative: 249
------------------------------------------------
nth 1:positive: 225 negative: 249
------------------------------------------------
nth 2:positive: 200 negative: 249
------------------------------------------------
nth 3:positive: 175 negative: 249
------------------------------------------------
nth 4:positive: 150 negative: 249
------------------------------------------------
nth 5:positive: 125 negative: 249
------------------------------------------------
nth 6:positive: 100 negative: 249
------------------------------------------------
nth 7:positive: 75 negative: 249
------------------------------------------------
nth 8:positive: 50 negative: 249
------------------------------------------------
nth 9:positive: 25 negative: 249
------------------------------------------------
Text(0.5, 1.0, 'decreasing positive sample')
```





Exercise 2 (4%):

Does the AUROC (Precision vs Recall), F1 score affected by imbalanced class?

```
# Your answer here
```

```
### YES, the AUROC and F1 score are affected by imbalanced class.
### Judging from the graph, the values of both the F1 score and
### the AUC are decreasing as the skewness ratio is getting larger
### (less positive samples). Hence, from the look of the graph,
### the AUROC and F1 score is affected by imbalanced class.
```

Let's go back to power point - slide 13

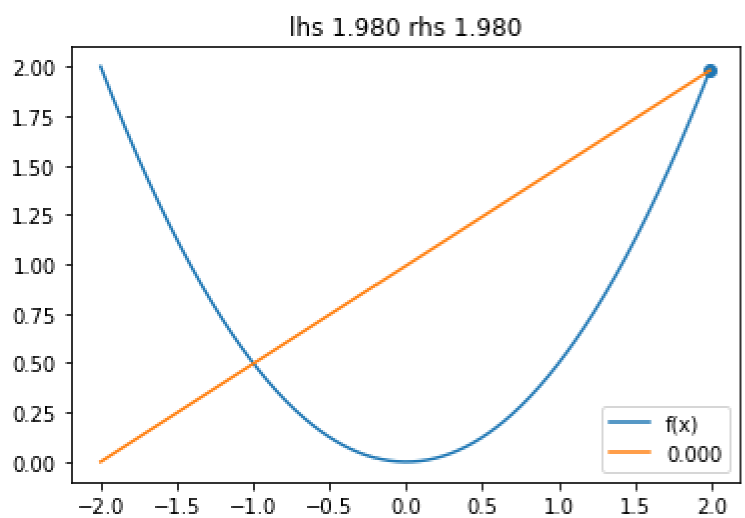
Convex function

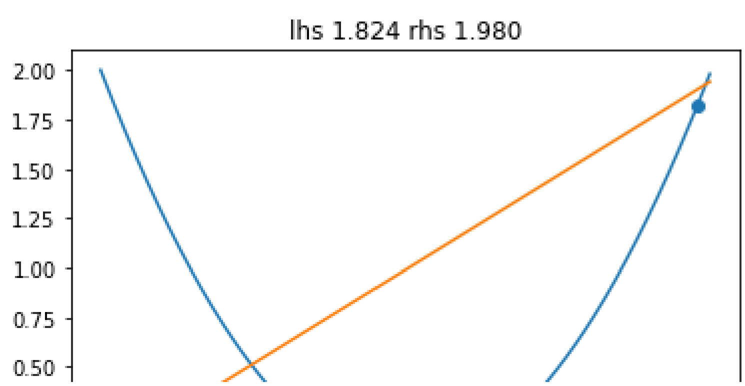
This is the code to generate the graph in slide 38

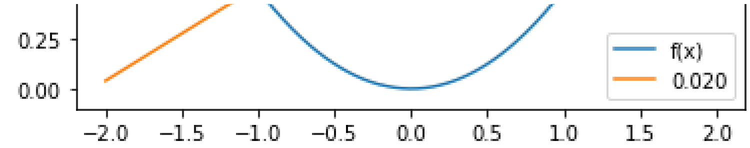
```
import numpy as np
import matplotlib.pyplot as plt
import imageio

x = np.arange(-2, 2, 0.01)

# choose one function to try
f = lambda x: 0.5 * x ** 2 # Convex
# f = lambda x: np.cos(np.pi * x)   # Nonconvex
# f = lambda x: -0.5 * x ** 4   # Nonconvex

filenames=[]
for lamda in np.arange(0, 1, 0.02):
    # LHS
    tmp_x = lamda*x[0] + (1-lamda)*x[-1]

    # RHS
    x_line, y_line = np.array([x[0], x[-1]]), np.array([lamda*f(x[0]), (1-lamda)*f(x[-1])])

    # compute LHS and RHS
    LHS = f(tmp_x)
    RHS = lamda*f(x[0]) + (1-lamda)*f(x[-1])
    if LHS > RHS:
        print('At lamda %0.3f, it is concave' % lamda)
        print('lhs %.5f rhs %.5f' % (LHS, RHS))

    plt.figure()
    # original graph
    plt.plot(x, f(x), label='f(x)')
    # plot RHS
    plt.plot(x_line, y_line, label='%0.3f' % lamda)
    # plot LHS
    plt.scatter(tmp_x, f(tmp_x))
```

```
    #title, legennd
    plt.title('lhs %.3f rhs %.3f' % (LHS, RHS))
    plt.legend()
    plt.savefig('lamda %0.3f.png' % lamda)
    # plt.close()
    filenames.append('lamda %0.3f.png' % lamda)

# Build GIF
with imageio.get_writer('mygif.gif', mode='I') as writer:
        for filename in filenames:
                image = imageio.imread(filename)
                writer.append_data(image)
```

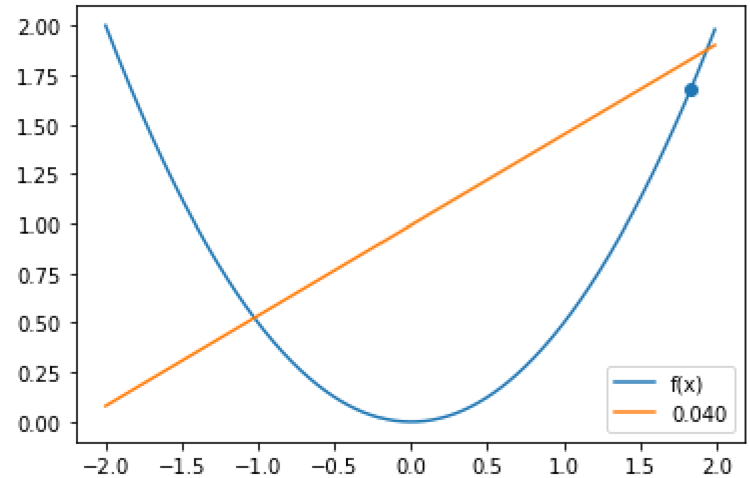
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:27: RuntimeWarning: More th
```



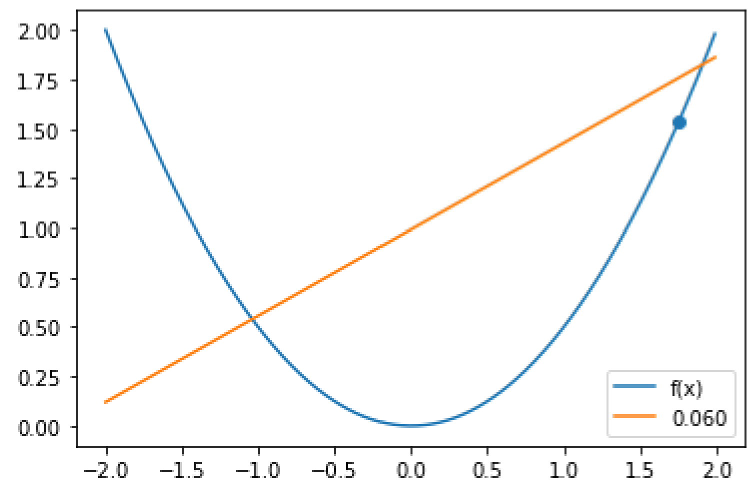




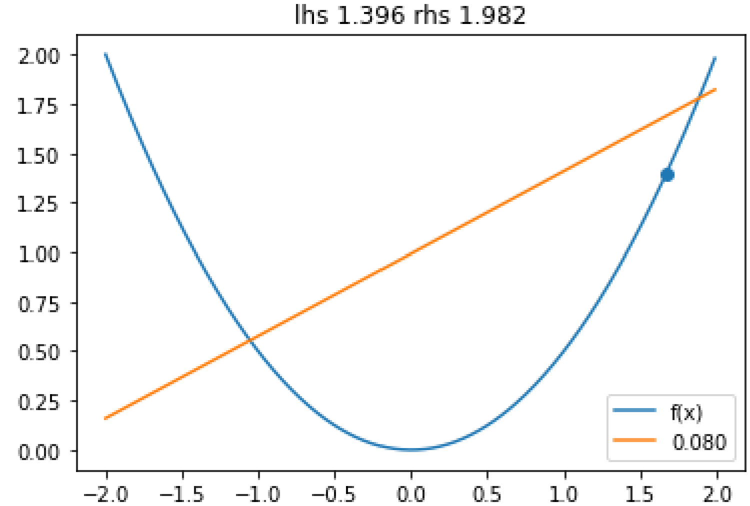
lhs 1.675 rhs 1.981



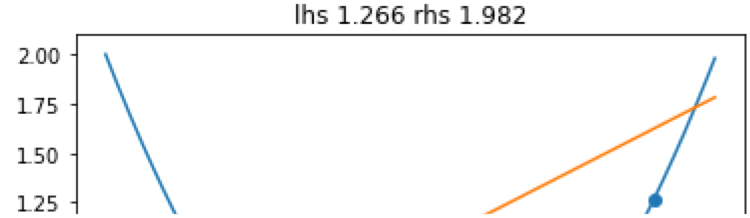
lhs 1.532 rhs 1.981

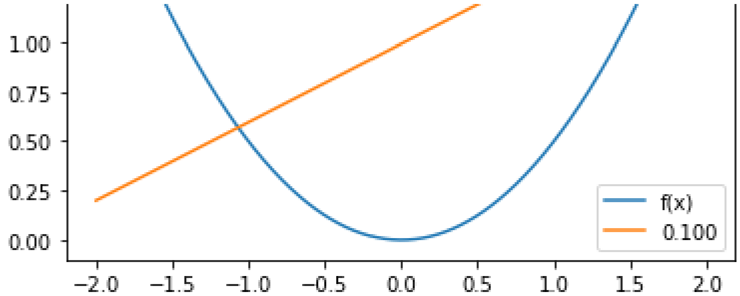


lhs 1.396 rhs 1.982

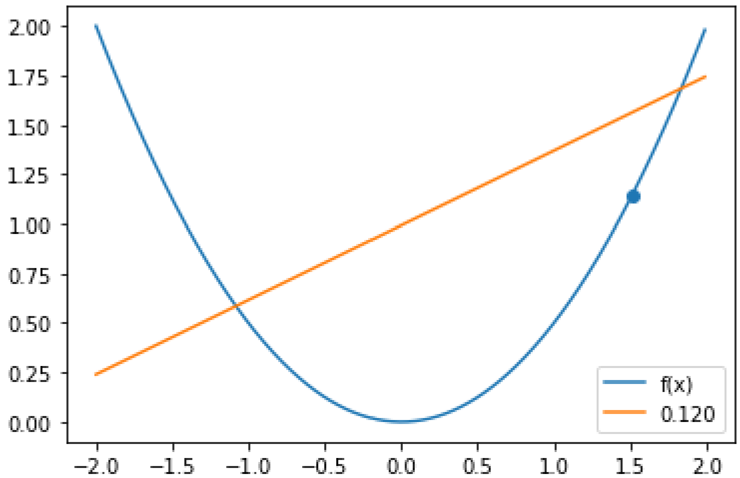


lhs 1.266 rhs 1.982

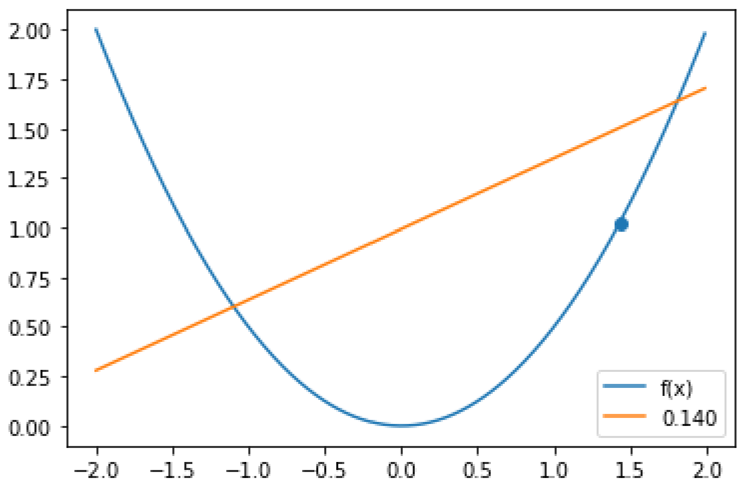




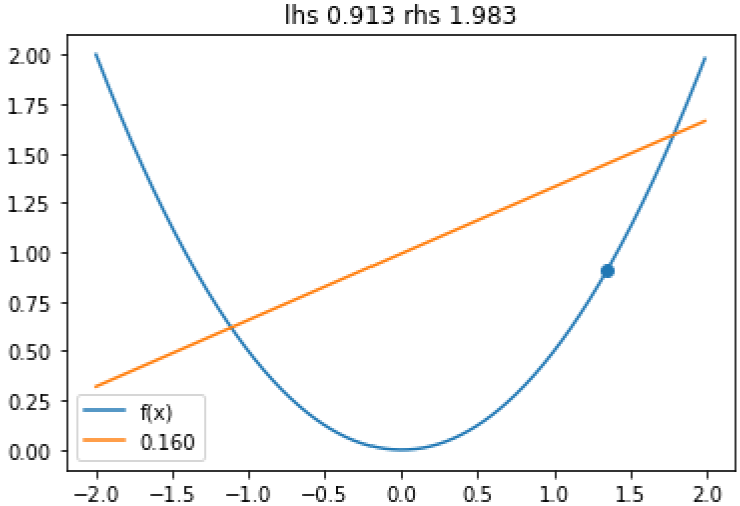
lhs 1.142 rhs 1.982



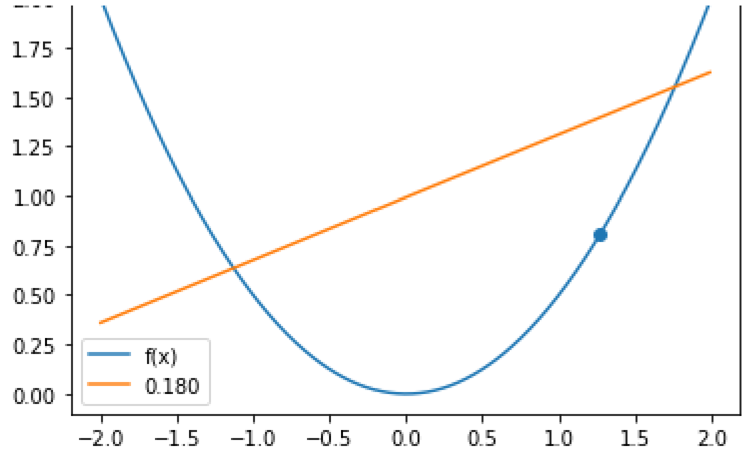
lhs 1.024 rhs 1.983



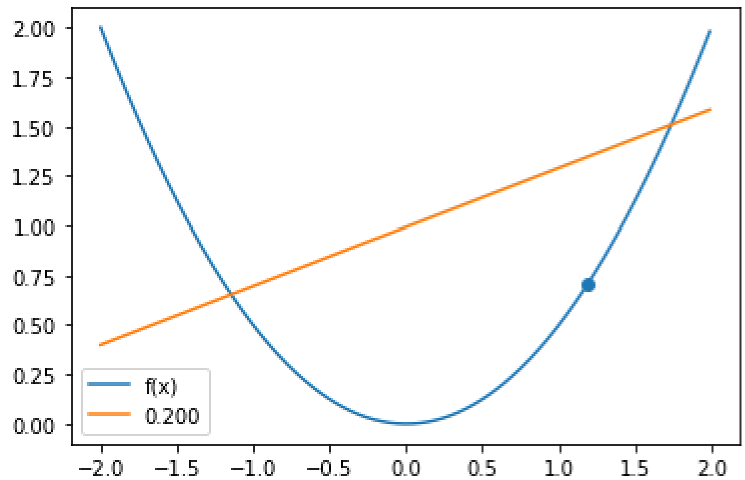
lhs 0.913 rhs 1.983



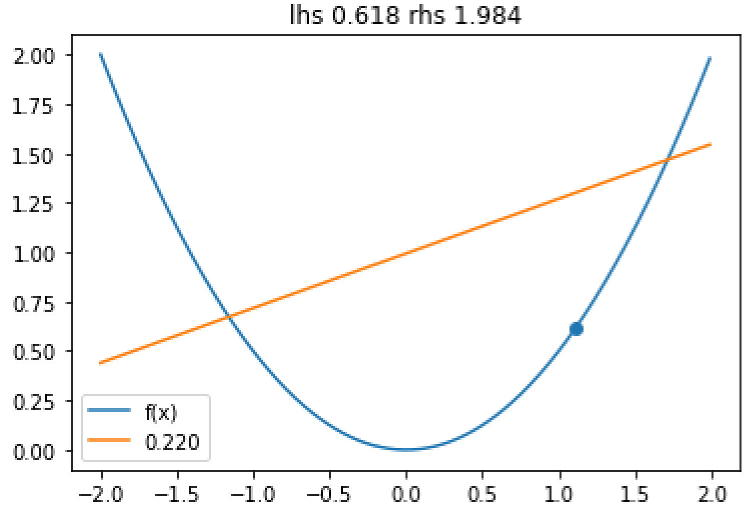
lhs 0.809 rhs 1.984



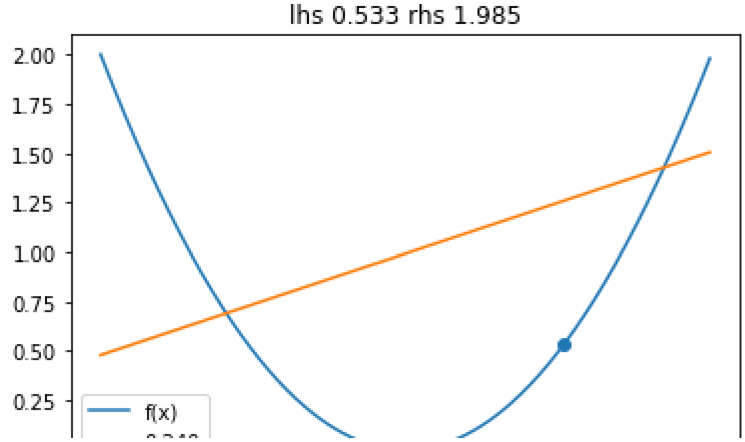
lhs 0.710 rhs 1.984



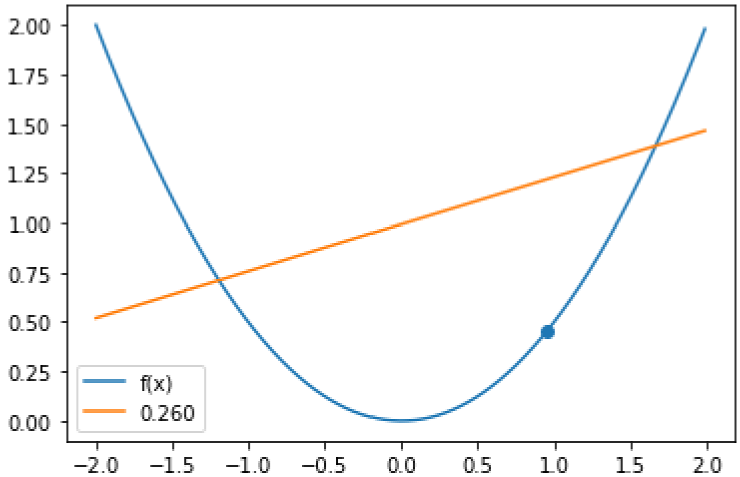
lhs 0.618 rhs 1.984



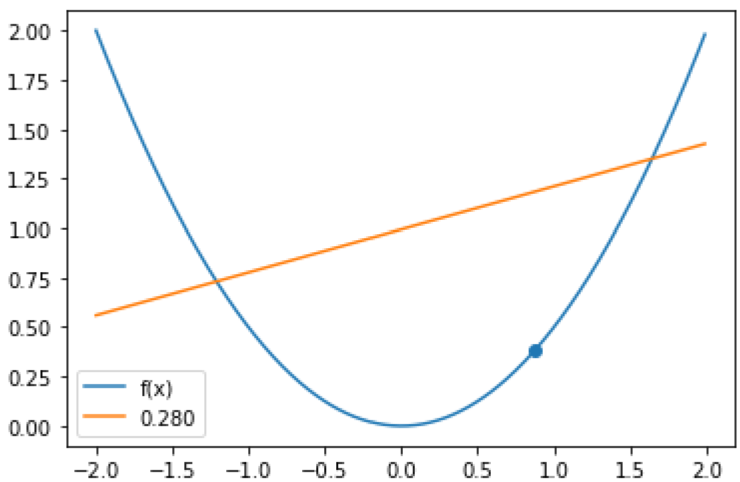
lhs 0.533 rhs 1.985



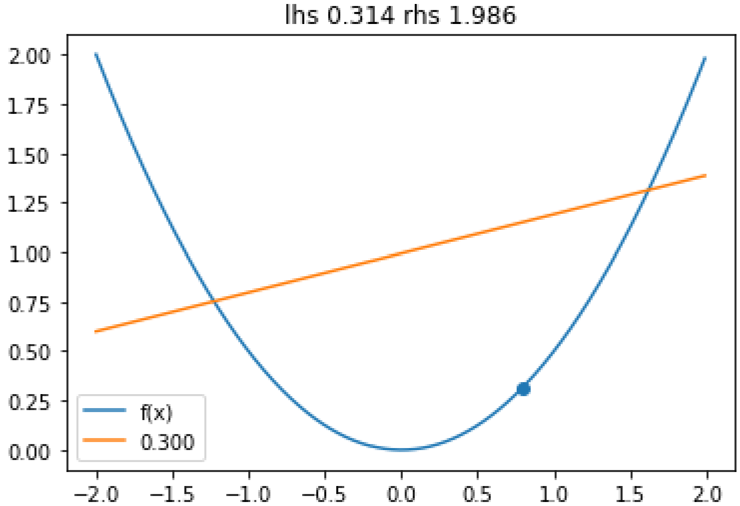
lhs 0.454 rhs 1.985



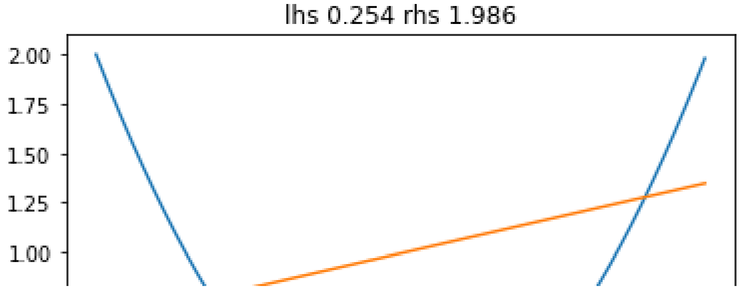
lhs 0.381 rhs 1.986

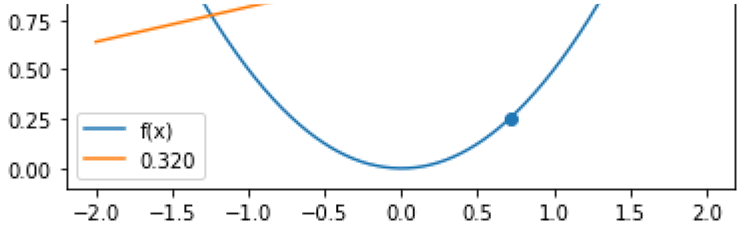


lhs 0.314 rhs 1.986

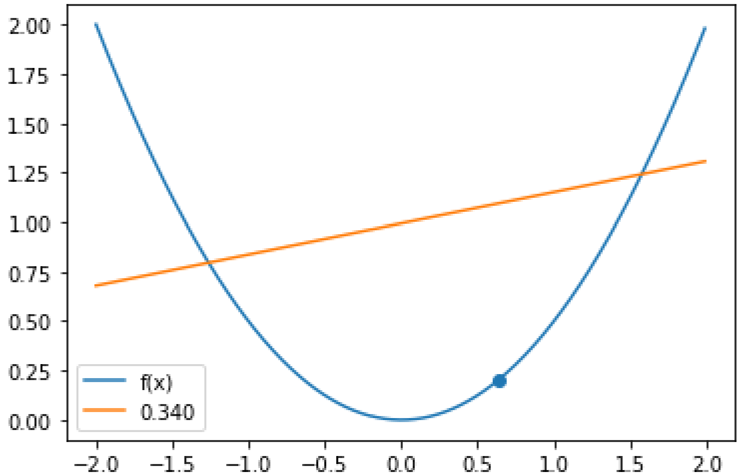


lhs 0.254 rhs 1.986

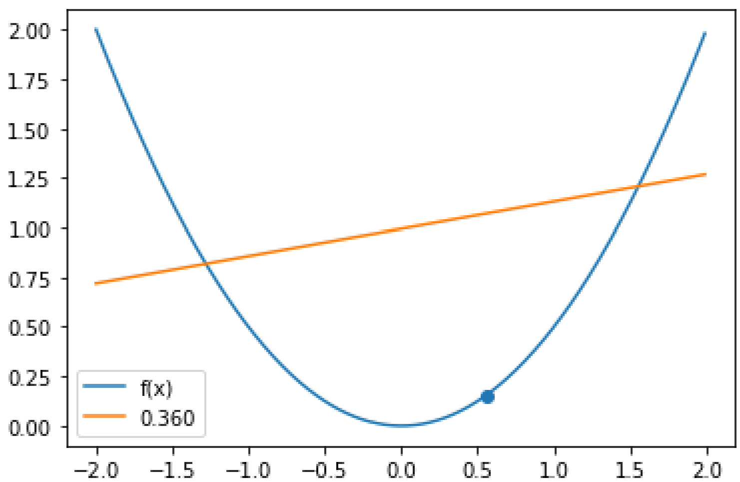




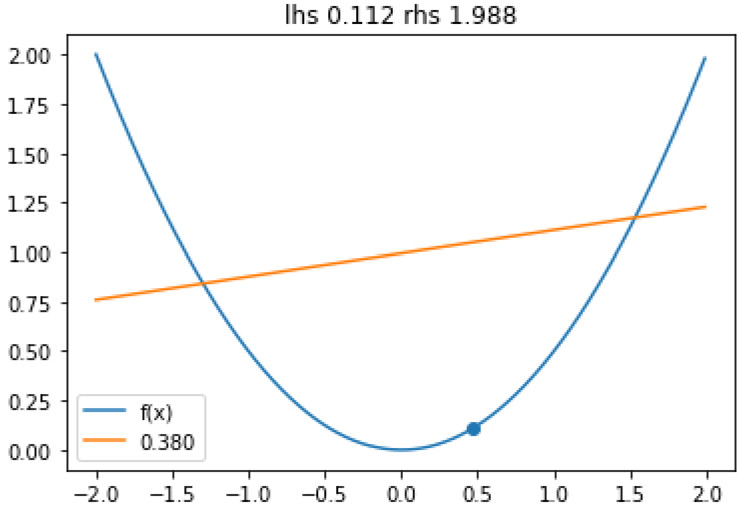
lhs 0.201 rhs 1.987



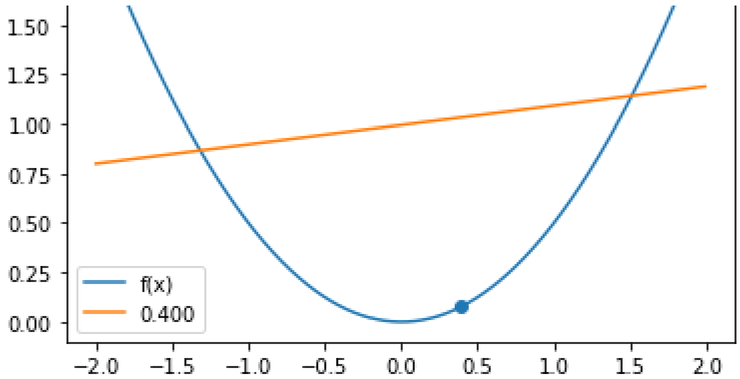
lhs 0.153 rhs 1.987

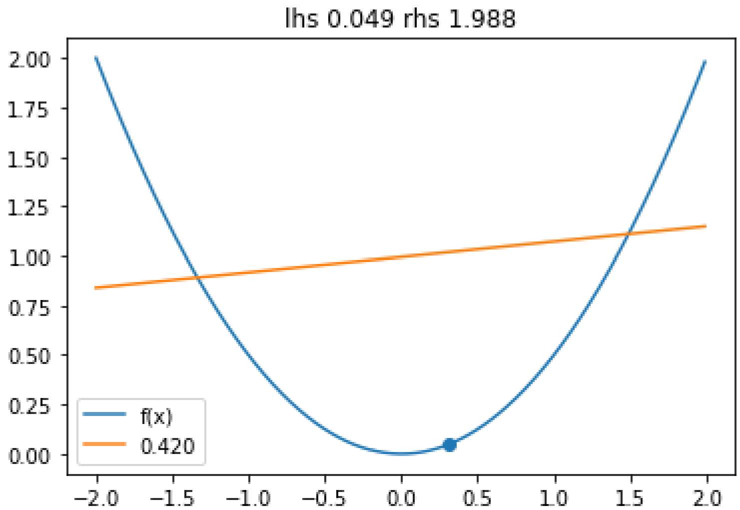


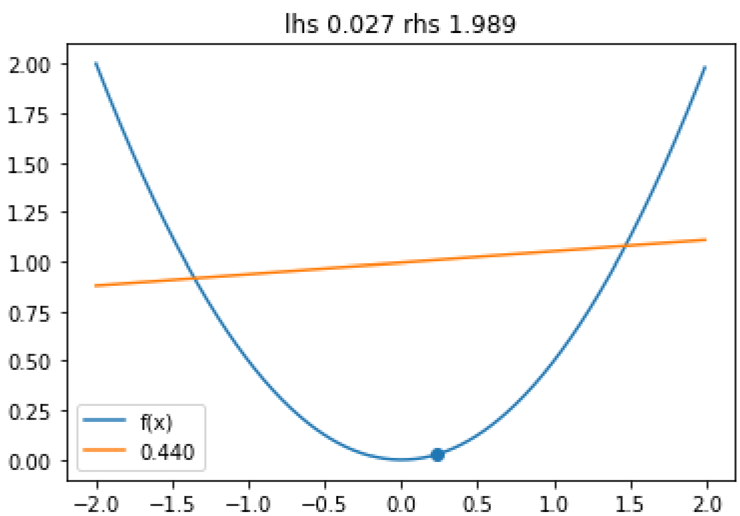
lhs 0.112 rhs 1.988

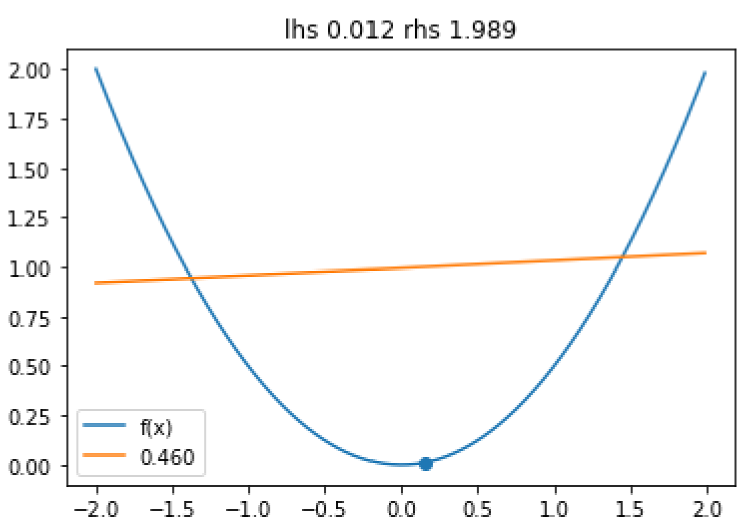


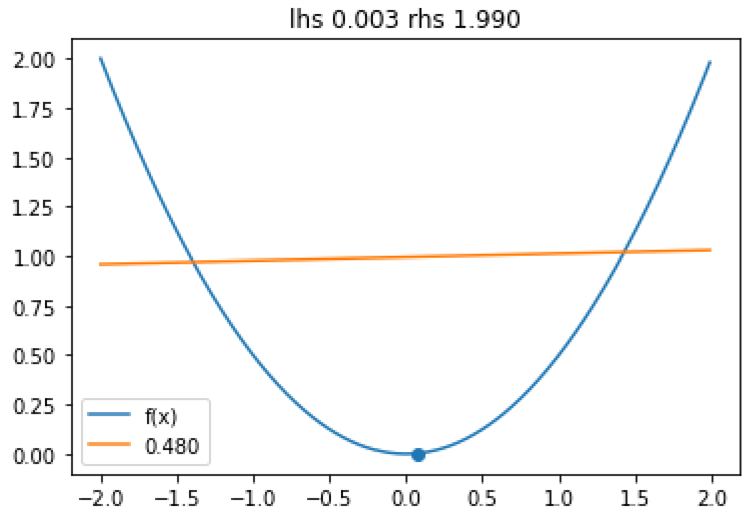
lhs 0.078 rhs 1.988

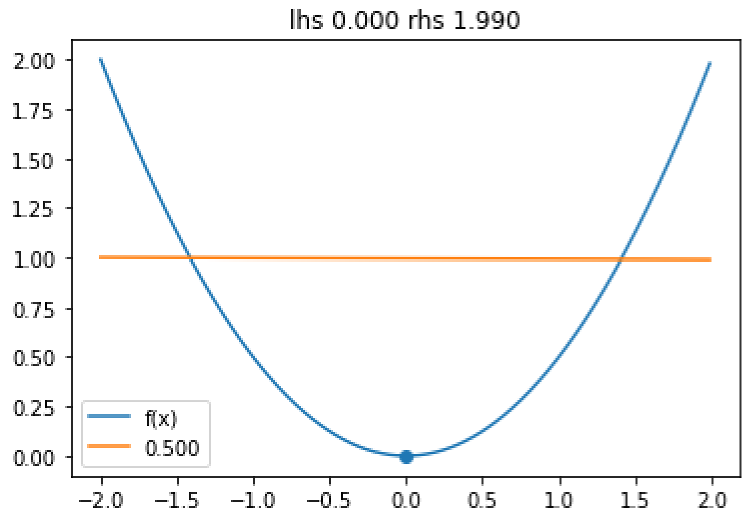

f(x)
0.400

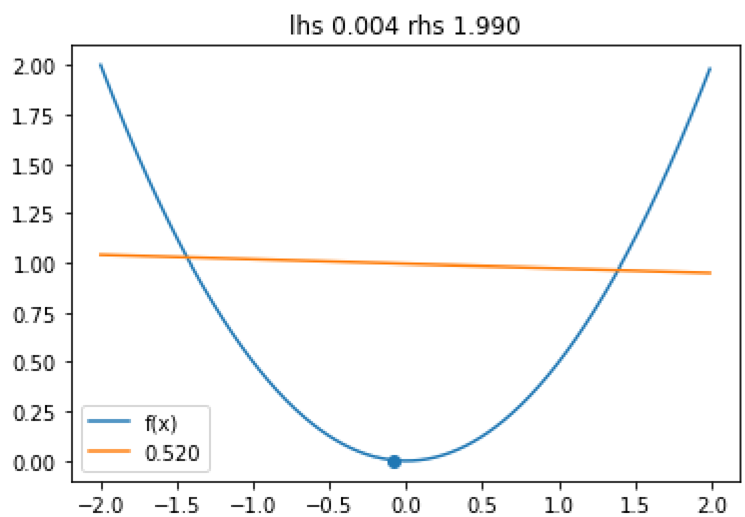

lhs 0.049 rhs 1.988
f(x)
0.420

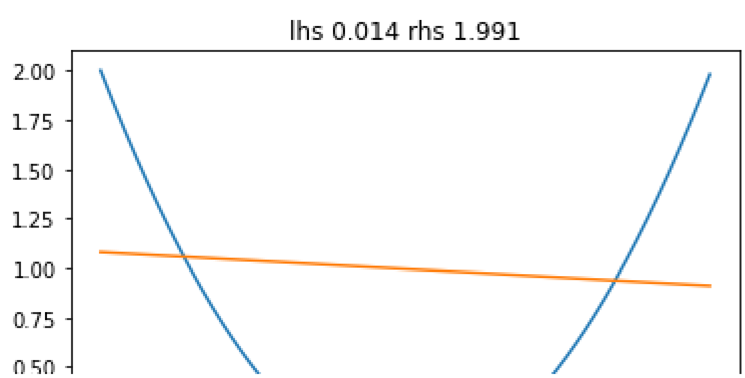

lhs 0.027 rhs 1.989
f(x)
0.440

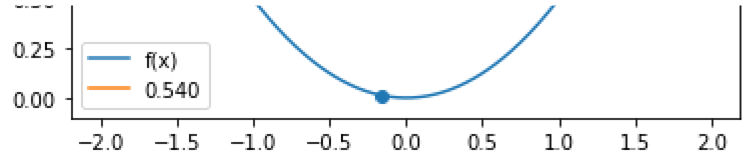

lhs 0.012 rhs 1.989
f(x)
0.460


lhs 0.003 rhs 1.990
f(x)
0.480

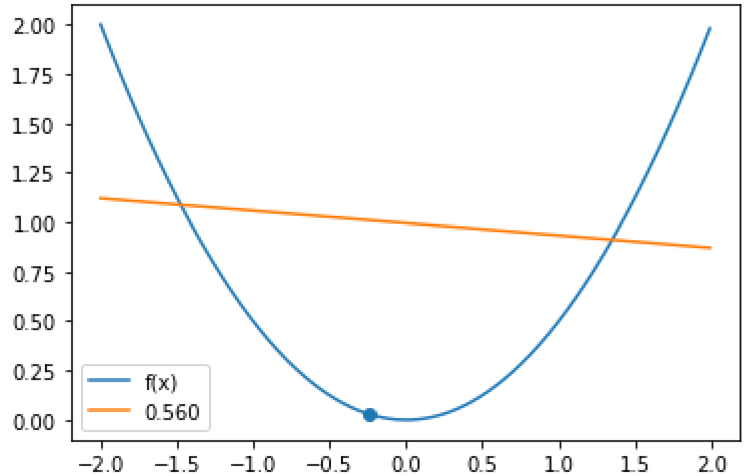

lhs 0.000 rhs 1.990
f(x)
0.500


lhs 0.004 rhs 1.990
f(x)
0.520

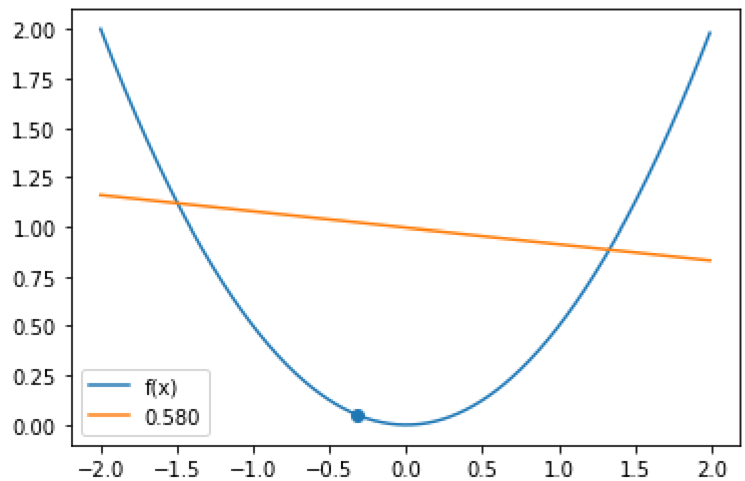

lhs 0.014 rhs 1.991



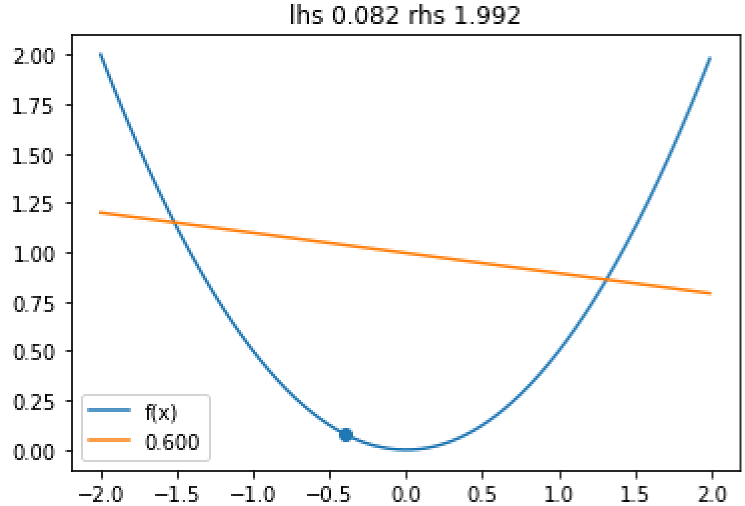
lhs 0.030 rhs 1.991



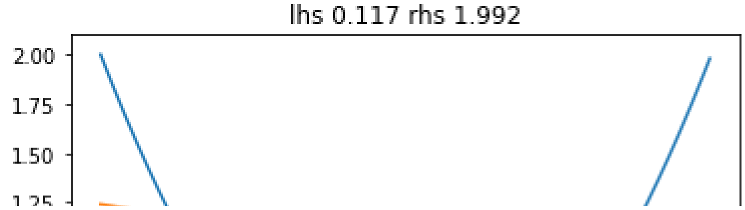
lhs 0.053 rhs 1.992

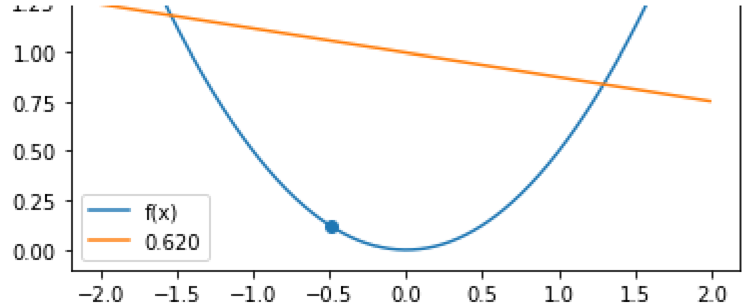


lhs 0.082 rhs 1.992

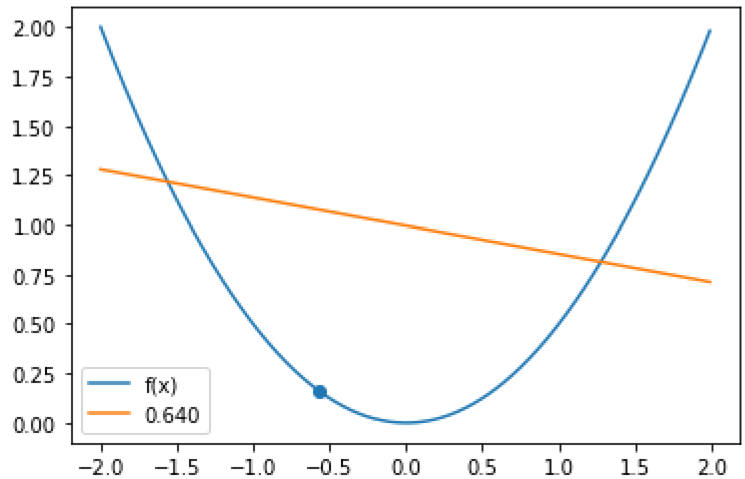


lhs 0.117 rhs 1.992

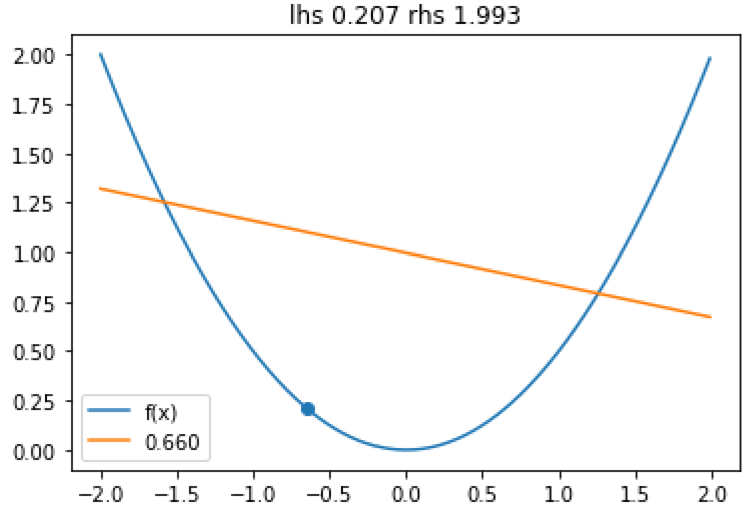




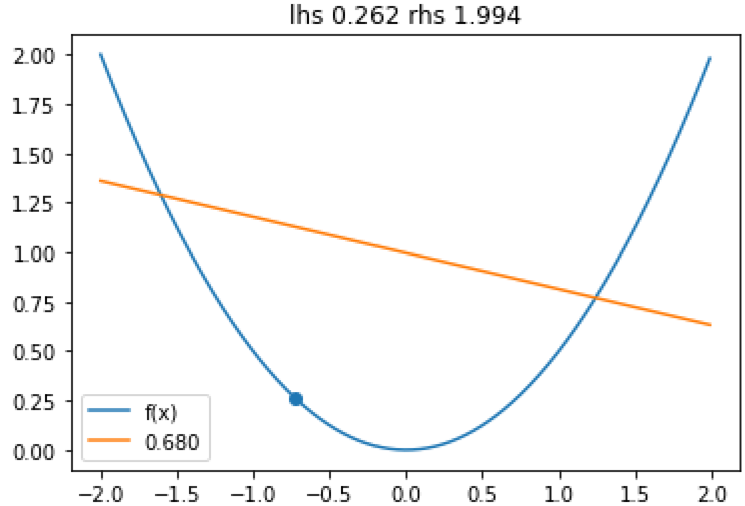
lhs 0.159 rhs 1.993



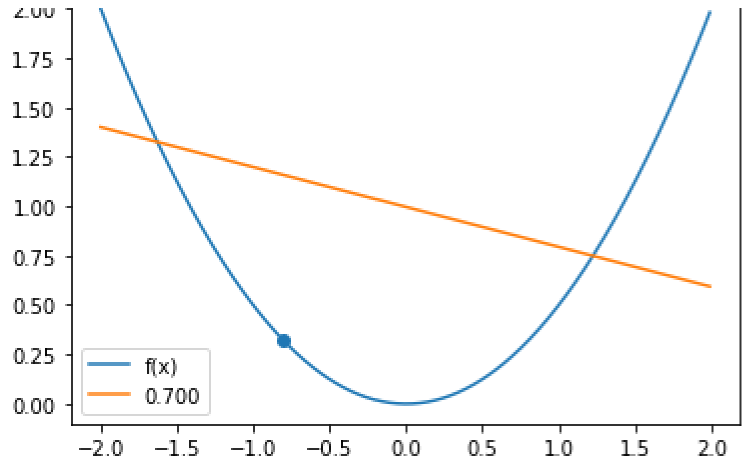
lhs 0.207 rhs 1.993



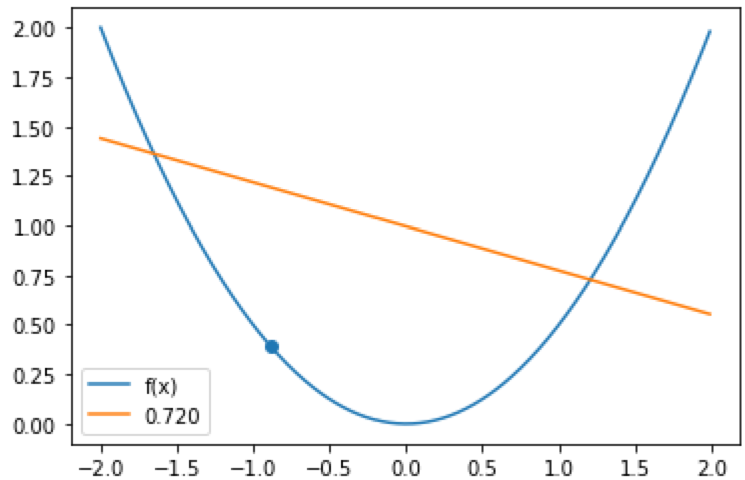
lhs 0.262 rhs 1.994



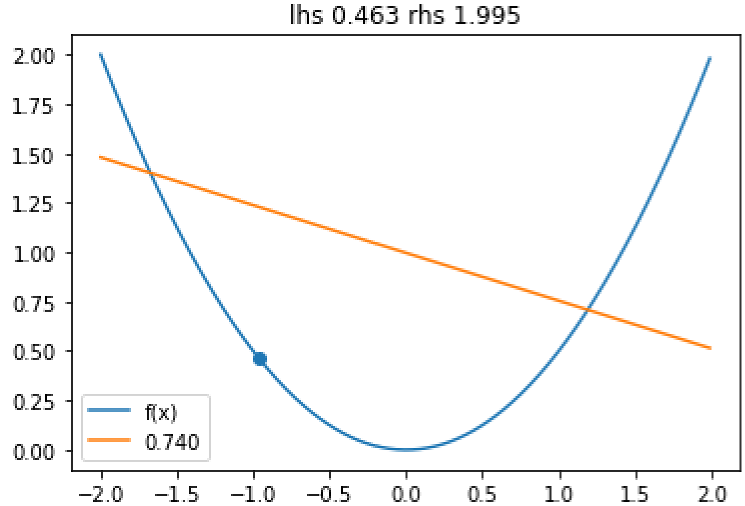
lhs 0.322 rhs 1.994



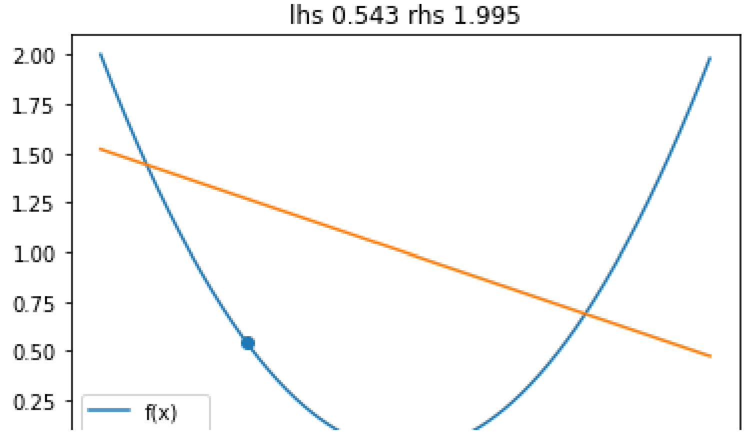
lhs 0.390 rhs 1.994

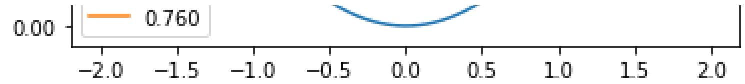


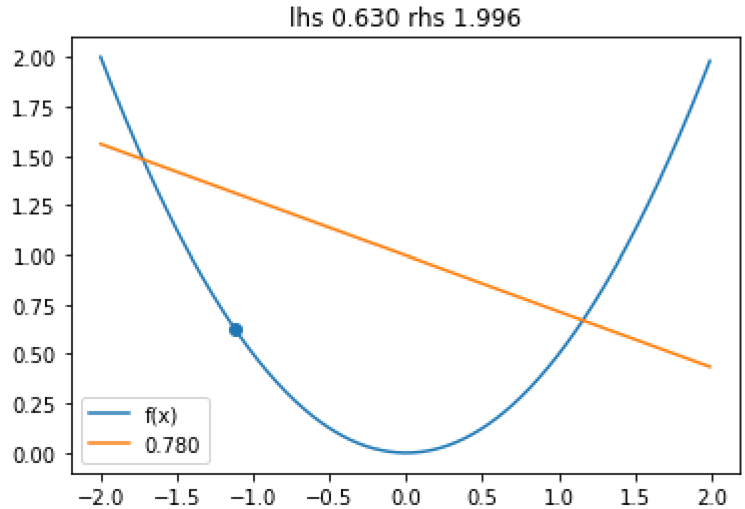
lhs 0.463 rhs 1.995

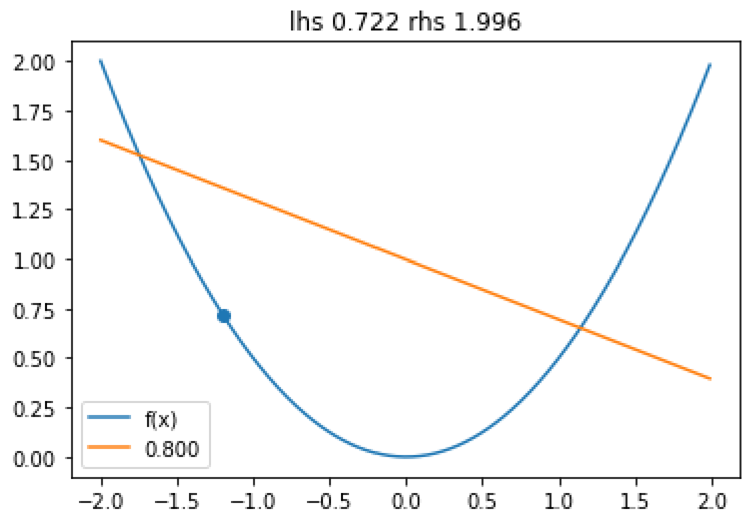


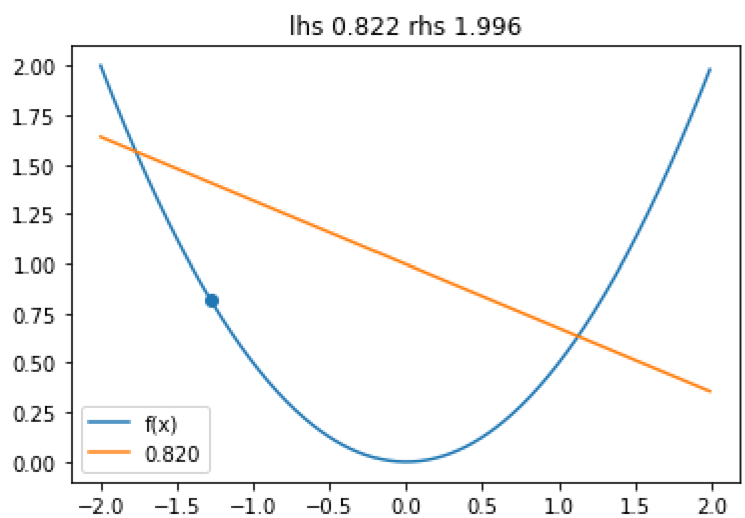
lhs 0.543 rhs 1.995

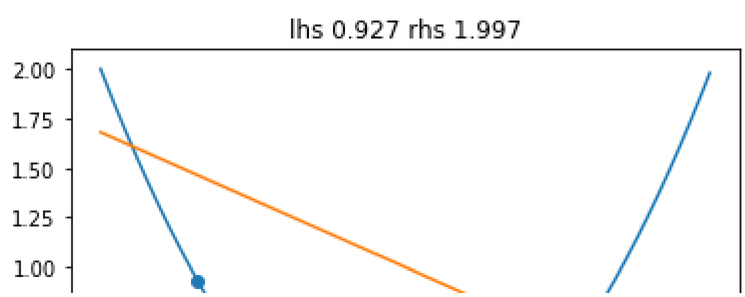


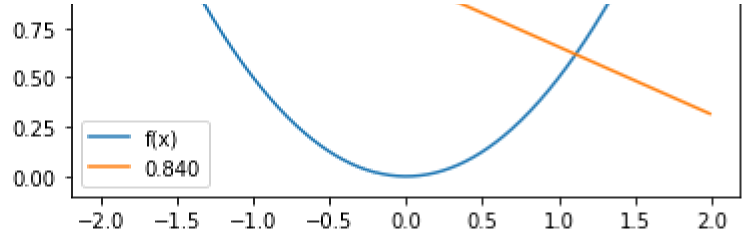

0.760


lhs 0.630 rhs 1.996
f(x)
0.780

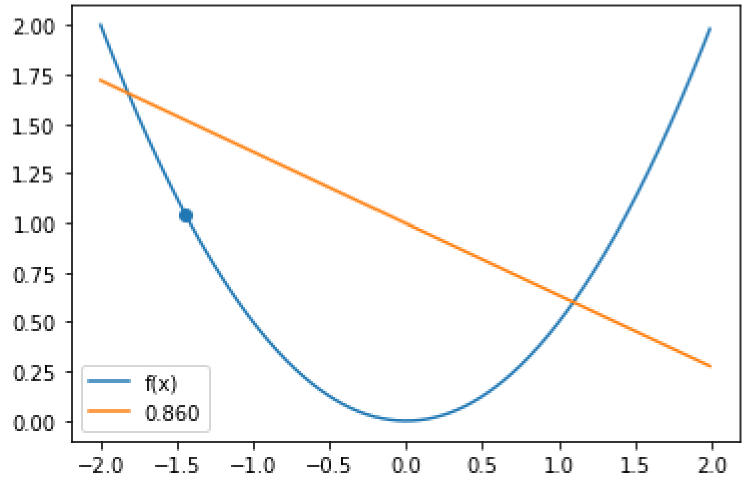

lhs 0.722 rhs 1.996
f(x)
0.800


lhs 0.822 rhs 1.996
f(x)
0.820

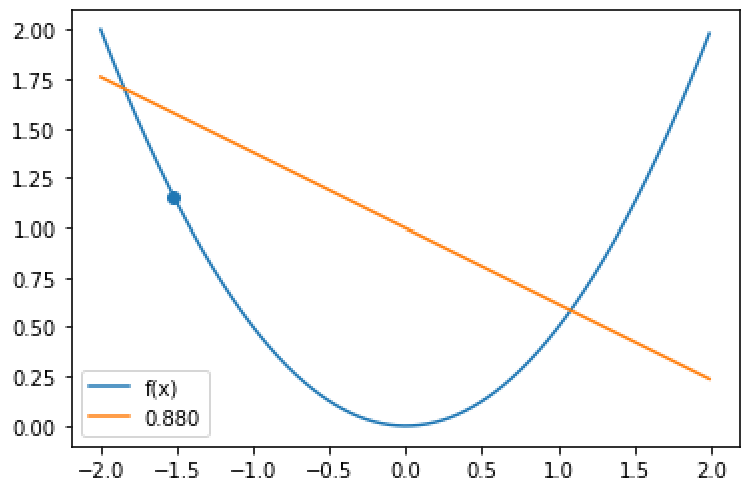

lhs 0.927 rhs 1.997



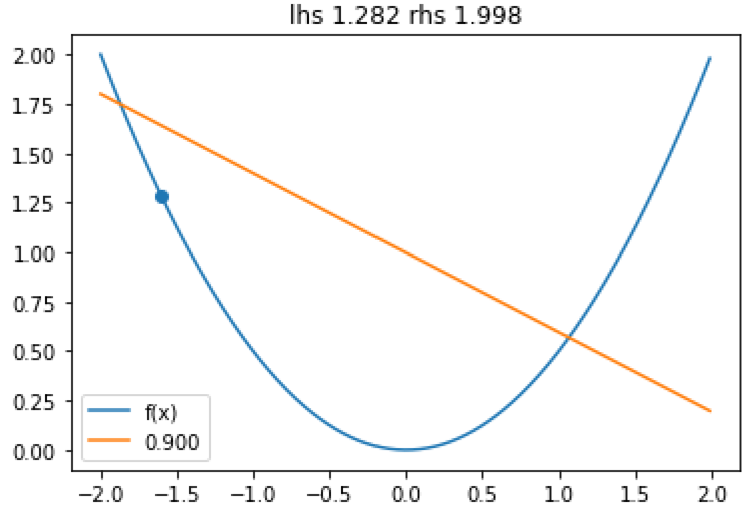
lhs 1.039 rhs 1.997



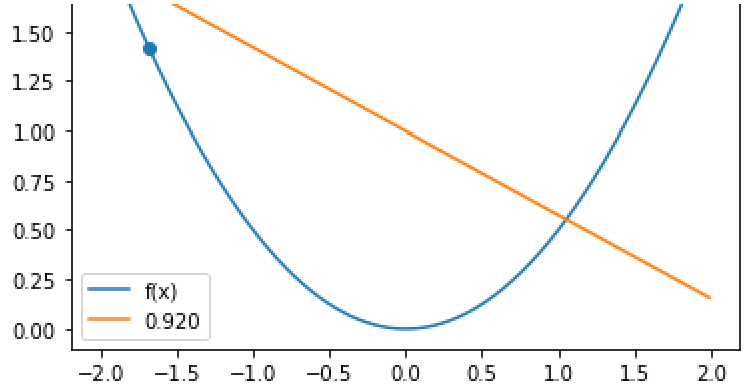
lhs 1.157 rhs 1.998

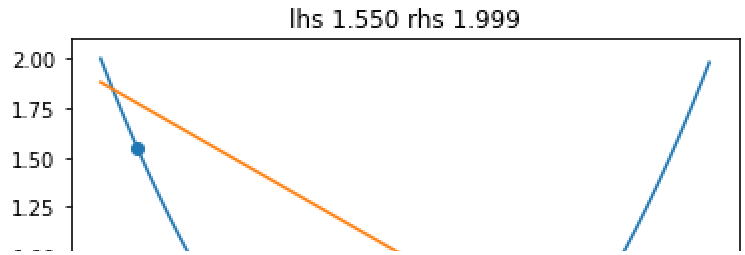


lhs 1.282 rhs 1.998



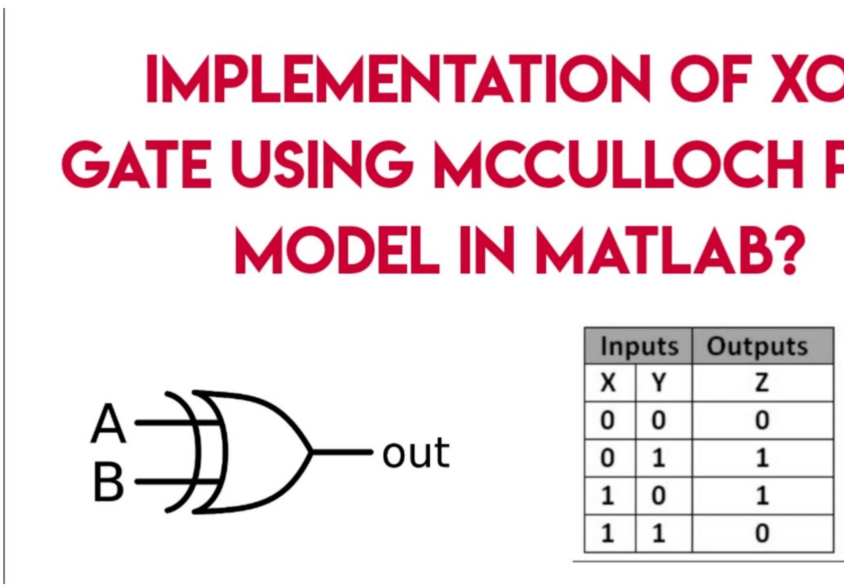
lhs 1.413 rhs 1.998





Understand how learning rate affects your SGD optimization

We will train a neural network for a pretty simple task, i.e. calculating the exclusive-or (XOR) of two input.



Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

```
import random
import numpy as np

# generate a function for XOR
x1 = random.randint(0, 1)
x2 = random.randint(0, 1)
yy = 0 if (x1 == x2) else 1
```

```
print('x1:', x1)
print('x2:',x2)
print('yy:',yy)
```

```
x1: 0
x2: 0
yy: 0
```

```
x1 = random.randint(0, 1)
x2 = random.randint(0, 1)
yy = 0 if (x1 == x2) else 1

# centered at zero
x1 = 2. * (x1 - 0.5)
x2 = 2. * (x2 - 0.5)
yy = 2. * (yy - 0.5)

print('x1:', x1)
print('x2:',x2)
print('yy:',yy)
```

```
x1: -1.0
x2: 1.0
yy: 1.0
```

```
x1 = random.randint(0, 1)
x2 = random.randint(0, 1)
yy = 0 if (x1 == x2) else 1

# centered at zero
x1 = 2. * (x1 - 0.5)
x2 = 2. * (x2 - 0.5)
yy = 2. * (yy - 0.5)

# add noise
x1 += 0.1 * random.random()
x2 += 0.1 * random.random()
yy += 0.1 * random.random()

print('x1:', x1)
print('x2:',x2)
print('yy:',yy)
```

```
x1: -0.9761001027430554
x2: -0.9773965308627811
yy: -0.9499307953442198
```

```
# make it into function
def make_data():
    x1 = random.randint(0, 1)
```

```
        x2 = random.randint(0, 1)
        yy = 0 if (x1 == x2) else 1

        # centered at zero
        x1 = 2. * (x1 - 0.5)
        x2 = 2. * (x2 - 0.5)
        yy = 2. * (yy - 0.5)

        # add noise
        x1 += 0.1 * random.random()
        x2 += 0.1 * random.random()
        yy += 0.1 * random.random()

        return [x1, x2, ], yy
```

```
# create batch samples
batch_size = 10
def make_batch():
        data = [make_data() for ii in range(batch_size)]
        labels = [label for xx, label in data]
        data = [xx for xx, label in data]
        return np.array(data, dtype='float32'), np.array(labels, dtype='float32')

print(make_batch())
```

```
(array([[ 1.0402569 , -0.9924554 ],
       [-0.91565394, -0.96305215],
       [-0.9073243 , -0.9211038 ],
       [-0.95045507,  1.0092024 ],
       [-0.92793477, -0.93937576],
       [-0.9921819 ,  1.0025389 ],
       [ 1.0699717 ,  1.0009911 ],
       [-0.9833375 , -0.9422065 ],
       [ 1.0168155 ,  1.0525709 ],
       [ 1.0665222 ,  1.0733525 ]], dtype=float32), array([ 1.0666791 , -0.9507392 , -0.9759766
        1.0191078 , -0.91856855, -0.9461869 , -0.9577108 , -0.96619433],
      dtype=float32))
```

```
# generate  500 train and 50 test data
train_data = [make_batch() for ii in range(500)]
test_data = [make_batch() for ii in range(50)]
```

```
# import torch libraries
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim
from torch.autograd import Variable



## Define our neural network class
torch.manual_seed(42)

class NN(nn.Module):
    def __init__(self):
        super(NN, self).__init__()

        self.dense1 = nn.Linear(2, 2)
        self.dense2 = nn.Linear(2, 1)

    def forward(self, x):
        x = F.tanh(self.dense1(x))
        x = self.dense2(x)
        return torch.squeeze(x)




# initialize our network
model = NN()
lr = 0.1
## optimizer = stochastic gradient descent
optimizer = optim.SGD(model.parameters(), lr)


## train and test functions

def train(epoch):#, lr = 0.001):
    #optimizer = optim.SGD(model.parameters(), lr)
    model.train()
    for batch_idx, (data, target) in enumerate(train_data):

        data, target = Variable(torch.from_numpy(data)), Variable(torch.from_numpy(target))
        optimizer.zero_grad()
        output = model(data)
        loss = F.mse_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} {}\tLoss: {:.4f}'.format(epoch, batch_idx * len(

def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_data:
        data, target = Variable(torch.from_numpy(data), volatile=True), Variable(torch.fr
```

```
            output = model(data)
            test_loss += F.mse_loss(output, target)
            correct += (np.around(output.data.numpy()) == np.around(target.data.numpy())).sum(

        test_loss /= len(test_data)
        test_loss = test_loss.item()

        print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
            test_loss, correct, batch_size * len(test_data), 100. * correct / (batch_size



## run experiment
nepochs = 200
#lr = 0.0001

print('lr=', lr)
for epoch in range(1, nepochs + 1):
    train(epoch)
    print('---------------------------------------------')
    test()

  # everytime rerun this cell, please re initialize your network, and re run the train
```

```
Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)

Train Epoch: 174 0      Loss: 0.0006
Train Epoch: 174 1000   Loss: 0.0011
Train Epoch: 174 2000   Loss: 0.0009
Train Epoch: 174 3000   Loss: 0.0007
Train Epoch: 174 4000   Loss: 0.0009
---------------------------------------------

Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)

Train Epoch: 175 0      Loss: 0.0006
Train Epoch: 175 1000   Loss: 0.0011
Train Epoch: 175 2000   Loss: 0.0009
Train Epoch: 175 3000   Loss: 0.0007
Train Epoch: 175 4000   Loss: 0.0009
---------------------------------------------

Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)

Train Epoch: 176 0      Loss: 0.0006
Train Epoch: 176 1000   Loss: 0.0011
Train Epoch: 176 2000   Loss: 0.0009
Train Epoch: 176 3000   Loss: 0.0007
Train Epoch: 176 4000   Loss: 0.0009
---------------------------------------------

Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)
```

```
Train Epoch: 177 0      Loss: 0.0006
Train Epoch: 177 1000   Loss: 0.0011
Train Epoch: 177 2000   Loss: 0.0009
Train Epoch: 177 3000   Loss: 0.0007
Train Epoch: 177 4000   Loss: 0.0009
---------------------------------------------

Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)

Train Epoch: 178 0      Loss: 0.0006
Train Epoch: 178 1000   Loss: 0.0011
Train Epoch: 178 2000   Loss: 0.0009
Train Epoch: 178 3000   Loss: 0.0007
Train Epoch: 178 4000   Loss: 0.0009
---------------------------------------------

Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)

Train Epoch: 179 0      Loss: 0.0006
Train Epoch: 179 1000   Loss: 0.0011
Train Epoch: 179 2000   Loss: 0.0009
Train Epoch: 179 3000   Loss: 0.0007
Train Epoch: 179 4000   Loss: 0.0009
---------------------------------------------

Test set: Average loss: 0.0009, Accuracy: 500/500 (100.00%)
```

Exercise 3 (6%)

For this experiment, try the following learning rate (lr=0.0001, 0.001, 0.01, 0.1). What do you observed?

For example, at lr=0.001, test acc reach 100% at epoch xx... At lr=0.001, test acc reach 100% at epoch xx. As lr increases / decreases, what happen?

Your answer here

With Learning Rate of 0.0001, the accuracy reached 100% at epoch 180.

With Learning Rate of 0.001, the accuracy reached 100% at epoch 20.

With Learning Rate of 0.01, the accuracy reached 100% at epoch 2.

With Learning Rate of 0.1, the accuracy reached 100% at epoch 1.

So we from the experiments we can see that as the learning rate increases, the time/epoch required for the model to reach 100% test accuracy is reduced, so larger the learning rate, faster the model reaches 100% test accuracy.

Submission Instructions

Once you are finished, follow these steps:

Restart the kernel and re-run this notebook from beginning to end by going to Kernel > Restart Kernel and Run All Cells. If this process stops halfway through, that means there was an error. Correct the error and repeat Step 1 until the notebook runs from beginning to end. Double check that there is a number next to each code cell and that these numbers are in order. Then, submit your lab as follows:

Go to File > Print > Save as PDF. Double check that the entire notebook, from beginning to end, is in this PDF file. Make sure Solution for Exercise 5 are in for marks. Upload the PDF to Spectrum.

Acknowledgement

Some of the works are inspired from

1. Effect of learning rate on AI model = https://www.commonlounge.com/discussion/5076b2cfb2364594ba608fca3ac606bb

✓ 31 秒 完成时间：20:39