In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Part 1

Note: Questions 1 and 3 will be first as they are in code format, 2 and 4 can be found in writing below.

Q1

In [ ]:

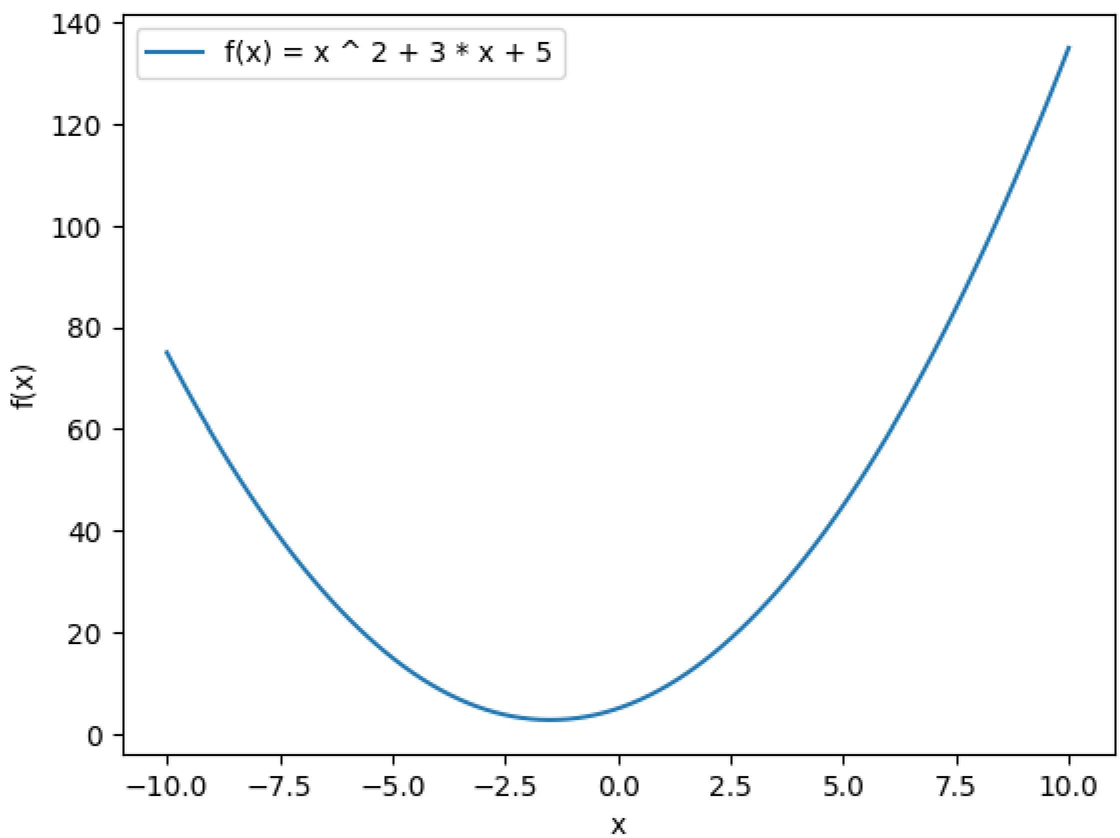
```
def f(x):
    x = float(x)
    return x ** 2 + 3 * x + 5
```

In [ ]:

```
x_list = np.linspace(-10, 10, int(1e2))
y_list = list(map(lambda x: f(x), x_list))
plt.plot(x_list, y_list, label= f"f(x) = x ^ 2 + 3 * x + 5")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.show
```

Out[ ]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Q2

In [ ]:

```
def grad_f(x):
    x = float(x)
    return 2*x + 3
```

Q3:

the extremum of the function will appear in: -b/2a = -1.5

Q4:

In [ ]:

```
def grad_update(grad, x, eta):
    x = float(x)
    return x - eta * grad(x)
```

Q5:

In [ ]:

```
def gradiant_decsent(grad_update, eta=0.05, epsilon=0.00001, lim = 12345, x0 = -10, x1 = -9):
    track_list = []
    while abs(x0 - x1) > epsilon and len(track_list) < lim:
        x0 = x1
        x1 = grad_update(grad_f, x0, eta)
        track_list.append(x1)

    return (x1, track_list) if len(track_list) <= lim else (x0, track_list)
```

In [ ]:

```
x, steps = gradiant_decsent(grad_update)
print(x, len(steps))
```

```
-1.5000857535845462 108
```

we can see that the gradiant_decsent function found the minimum extrenum {{x}} point within delta of epsilon: {{epsilon}} from what we found as the gobal minimum being -5

as we discussed in the tutorials, there might be 2 reason for why this happens:

1 - the epsilon is too big, and it stops before it reaches the extremum

2 - the eta is too big, and it jums over the extremum

Q6:

In [ ]:

```
min_steps = 10000
res = [0,0,0,0]
for eta in [0.05, 0.06, 0.07]:
  for epsilon in [0.00001, 0.00005, 0.0001]:
    for x1 in [-9, -8, -7]:
      x, steps = gradiant_decsent(grad_update, eta=eta, epsilon=epsilon, x1=x1)
      print(f'Reached: {x} with a T: {len(steps)} (steps).\nWith the parameters: epsilon= {epsilor
      if len(steps) < min_steps:
        min_steps = len(steps)
        res[0] = x
        res[1] = epsilon
        res[2] = eta
        res[3] = x1
print(f'\nThe min T was: {min_steps}, reaching the x: {res[0]}\nParams: epsilon= {res[1]}, eta= {r
```

```
Reached: -1.5000857535845462 with a T: 108 (steps).
With the parameters: epsilon= 1e-05, eta= 0.05, x1=-9
Reached: -1.5000825775258593 with a T: 107 (steps).
With the parameters: epsilon= 1e-05, eta= 0.05, x1=-8
Reached: -1.5000862633223602 with a T: 105 (steps).
```

```
With the parameters: epsilon= 1e-05, eta= 0.05, x1=-7
Reached: -1.5004164996504408 with a T: 93 (steps).
With the parameters: epsilon= 5e-05, eta= 0.05, x1=-9
Reached: -1.5004456374860684 with a T: 91 (steps).
With the parameters: epsilon= 5e-05, eta= 0.05, x1=-8
Reached: -1.500418975414252 with a T: 90 (steps).
With the parameters: epsilon= 5e-05, eta= 0.05, x1=-7
Reached: -1.5008707973027648 with a T: 86 (steps).
With the parameters: epsilon= 0.0001, eta= 0.05, x1=-9
Reached: -1.5008385455508106 with a T: 85 (steps).
With the parameters: epsilon= 0.0001, eta= 0.05, x1=-8
Reached: -1.5008759735098685 with a T: 83 (steps).
With the parameters: epsilon= 0.0001, eta= 0.05, x1=-7
Reached: -1.5000665249304308 with a T: 91 (steps).
With the parameters: epsilon= 1e-05, eta= 0.06, x1=-9
Reached: -1.50006551697694 with a T: 90 (steps).
With the parameters: epsilon= 1e-05, eta= 0.06, x1=-8
Reached: -1.5000715876059219 with a T: 88 (steps).
With the parameters: epsilon= 1e-05, eta= 0.06, x1=-7
Reached: -1.5003505174883744 with a T: 78 (steps).
With the parameters: epsilon= 5e-05, eta= 0.06, x1=-9
Reached: -1.5003452066173386 with a T: 77 (steps).
With the parameters: epsilon= 5e-05, eta= 0.06, x1=-8
Reached: -1.5003319294397488 with a T: 76 (steps).
With the parameters: epsilon= 5e-05, eta= 0.06, x1=-7
Reached: -1.5006641961140044 with a T: 73 (steps).
With the parameters: epsilon= 0.0001, eta= 0.06, x1=-9
Reached: -1.5006541325365197 with a T: 72 (steps).
With the parameters: epsilon= 0.0001, eta= 0.06, x1=-8
Reached: -1.5007147427190992 with a T: 70 (steps).
With the parameters: epsilon= 0.0001, eta= 0.06, x1=-7
Reached: -1.500058336512234 with a T: 78 (steps).
With the parameters: epsilon= 1e-05, eta= 0.07, x1=-9
Reached: -1.500058788733259 with a T: 77 (steps).
With the parameters: epsilon= 1e-05, eta= 0.07, x1=-8
Reached: -1.5000578422241369 with a T: 76 (steps).
With the parameters: epsilon= 1e-05, eta= 0.07, x1=-7
Reached: -1.500306519068238 with a T: 67 (steps).
With the parameters: epsilon= 5e-05, eta= 0.07, x1=-9
Reached: -1.5002656498591396 with a T: 67 (steps).
With the parameters: epsilon= 5e-05, eta= 0.07, x1=-8
Reached: -1.5003039219173082 with a T: 65 (steps).
With the parameters: epsilon= 5e-05, eta= 0.07, x1=-7
Reached: -1.5005603555680742 with a T: 63 (steps).
With the parameters: epsilon= 0.0001, eta= 0.07, x1=-9
Reached: -1.500564699409687 with a T: 62 (steps).
With the parameters: epsilon= 0.0001, eta= 0.07, x1=-8
Reached: -1.5005556076481716 with a T: 61 (steps).
With the parameters: epsilon= 0.0001, eta= 0.07, x1=-7

The min T was: 61, reaching the x: -1.5005556076481716
Params: epsilon= 0.0001, eta= 0.07, x1= -7
```

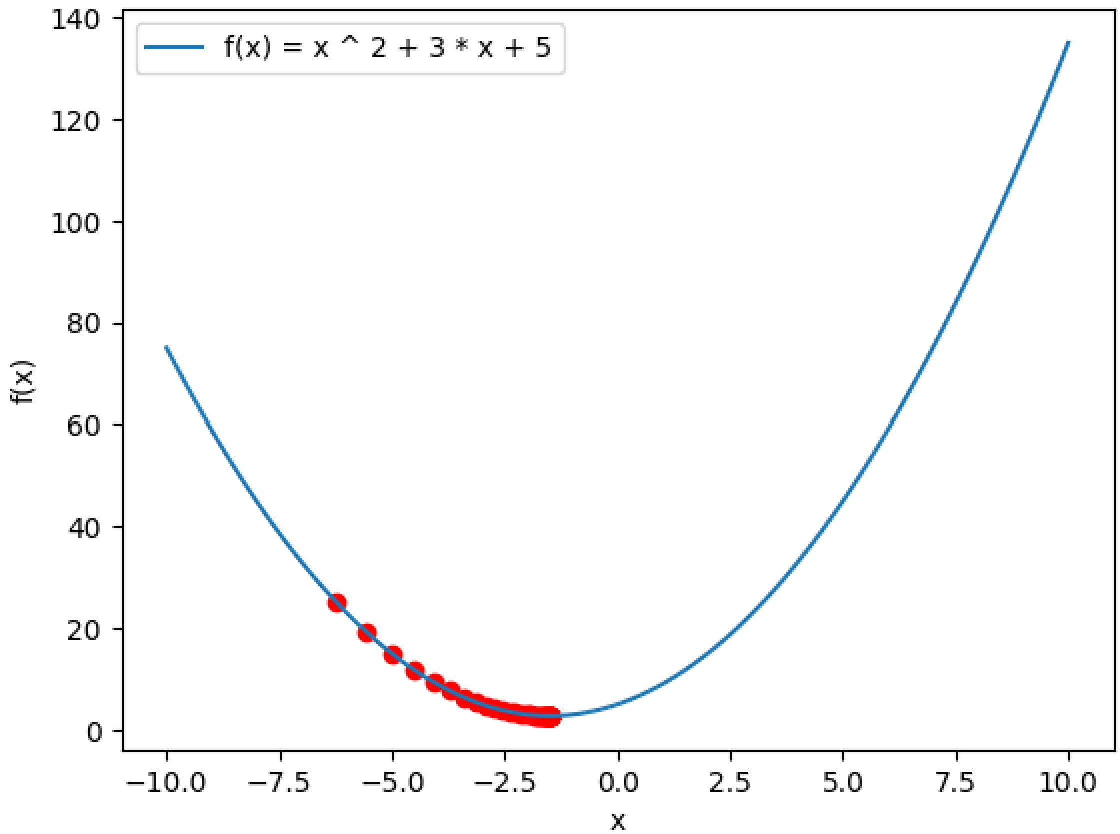
Q7:

In [ ]:

```
temp, steps = gradiant_decsent(grad_update, eta=res[2], epsilon=res[1], x1=res[3])
plt.plot(x_list, y_list, label= f"f(x) = x ^ 2 + 3 * x + 5")
plt.scatter(steps, list(f(x) for x in steps), c="r")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.show
```

Out[ ]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Part 2

Q4:

In [ ]:

```python
def sub_grad(x, y, w, b, d, lam):

  # just like we did in the questions before:
  if 1 - y*(np.dot(w,x) + b) > 0:
    sub_grad_w = -y * x + 2*lam*w
    sub_grad_b = -y
  else:
    sub_grad_w = 2*lam*w
    sub_grad_b = 0
  return sub_grad_w, sub_grad_b
```

In [ ]:

```python
def svm_with_sgd(x, y, lam=0, epochs=1000, l_rate=0.01, sgd_type='practical'):
  np.random.seed(2)
  m = x.shape[0]
  d = x.shape[1]
  w = np.random.uniform(size=d)
  b = np.random.uniform(size=1)

  if sgd_type == 'practical':
    for i in range(epochs):
      perm = np.random.permutation(m)
      for j in perm:
        sub_grad_w, sub_grad_b = sub_grad(x[j], y[j], w, b, d, lam)
        w = w - l_rate * sub_grad_w
```

```
        b = b - l_rate * sub_grad_b
    return w, b

  if sgd_type == 'theory':
    W = []
    B = 0
    for i in range(m*epochs):
      j = np.random.randint(m)
      sub_grad_w, sub_grad_b = sub_grad(x[j], y[j], w, b, d, lam)
      w = w - l_rate * sub_grad_w
      b = b - l_rate * sub_grad_b
      W.append(w)
      B += b
    W_np = np.array(W)
    return np.sum(W_np, axis= 0) / (m*epochs), B / (m*epochs)
```

Q5:

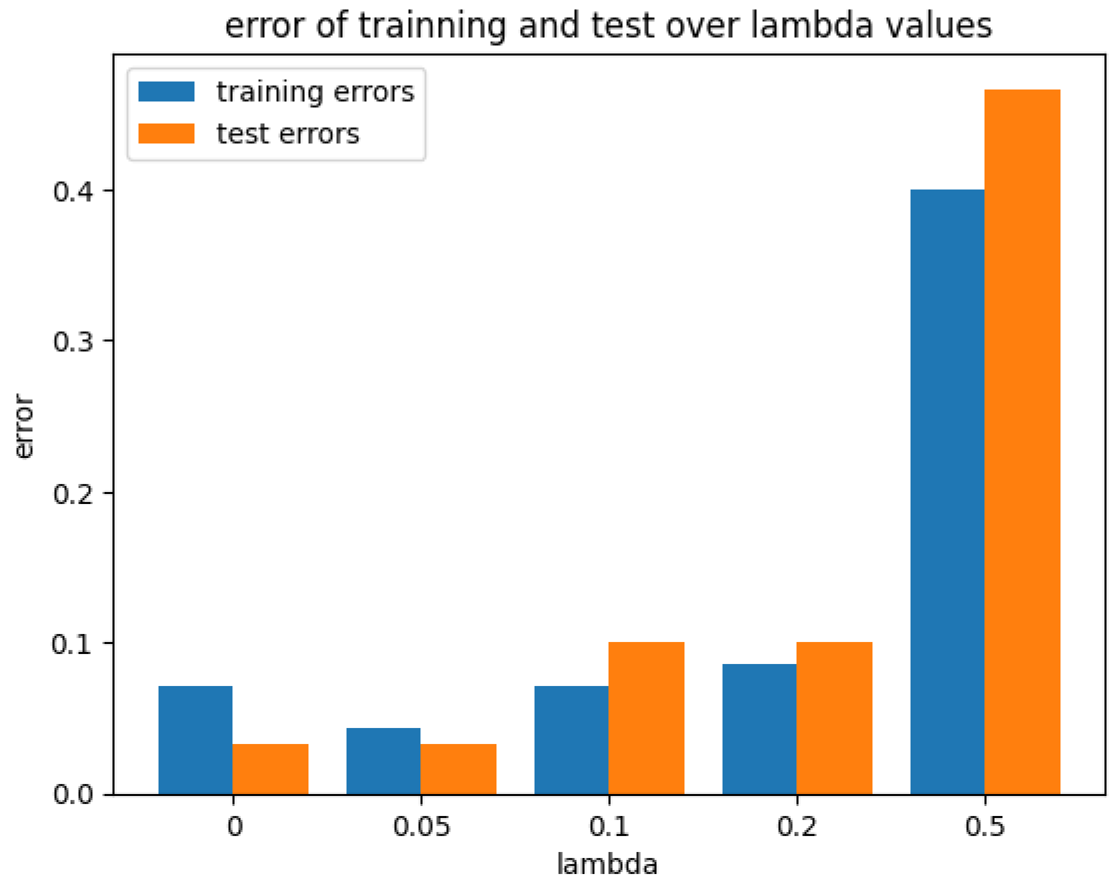
In [ ]:

```
def calculate_error(w, b, x, y):
  pred = np.zeros(x.shape[0])
  for i in range(len(pred)):
    pred[i] = 1 if np.dot(w,x[i]) + b > 0 else -1
  return np.mean(y != pred)
```

Q6:

In [ ]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X = X[y != 0]
y = y[y != 0]
y[y==2] = -1
X = X[:, 2:4]
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=0)

train_err_lst = []
test_err_lst = []
margin_lst = []

for lam in [0, 0.05, 0.1, 0.2, 0.5]:
  w_train, b_train = svm_with_sgd(X_train, y_train, lam)
  train_err = calculate_error(w_train, b_train, X_train, y_train)
  train_err_lst.append(train_err)
  test_err = calculate_error(w_train, b_train, X_val, y_val)
  test_err_lst.append(test_err)
  margin = 1/np.linalg.norm(w_train)
  margin_lst.append(margin)

bar = np.arange(5)
plt.bar(bar - 0.2, train_err_lst, label = 'training errors', align= 'center', width= 0.4)
plt.bar(bar + 0.2, test_err_lst, label = 'test errors', align= 'center', width= 0.4)
plt.title('error of trainning and test over lambda values')
plt.xlabel('lambda')
plt.ylabel('error')
plt.xticks(bar, [0,0.05,0.1, 0.2, 0.5])
plt.legend()
plt.show()
```

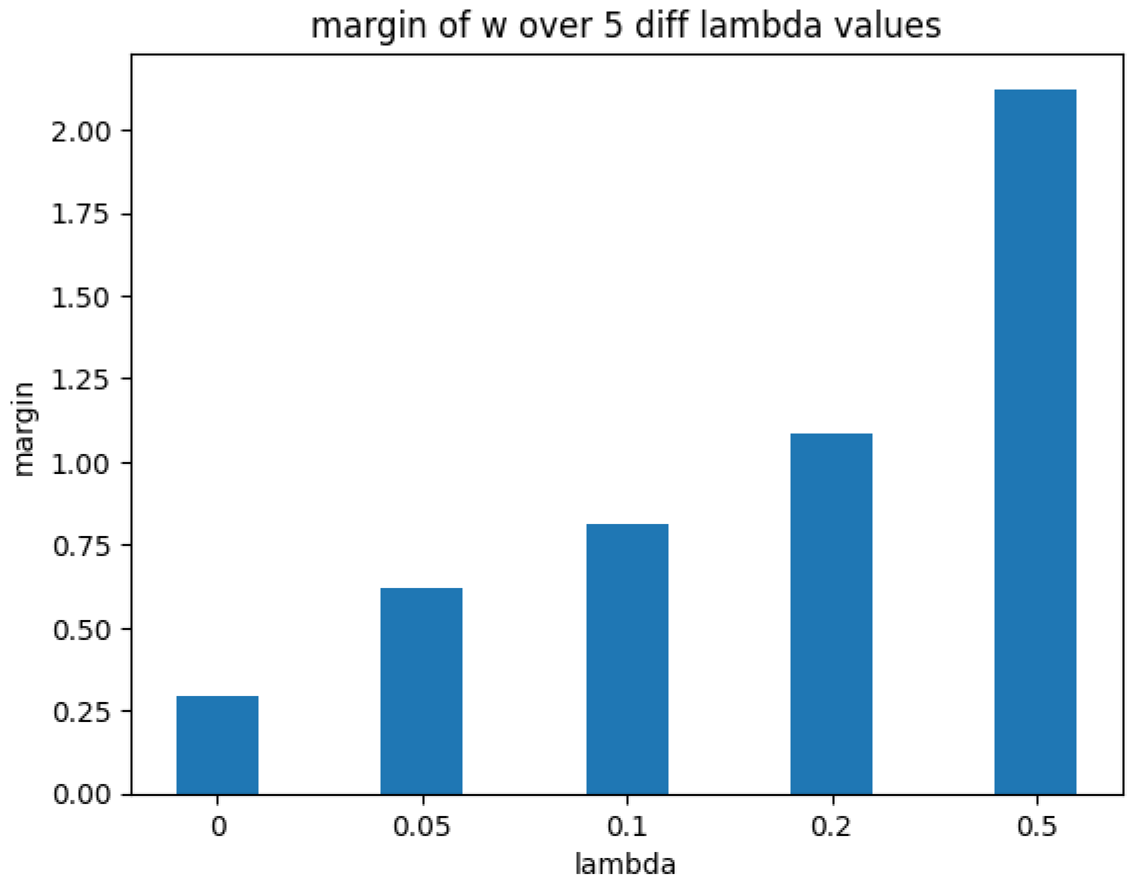


In [ ]:

```
plt.bar(bar, margin_lst, align= 'center', width= 0.4)
plt.title('margin of w over 5 diff lambda values')
plt.xticks(bar, [0,0.05,0.1, 0.2, 0.5])
plt.xlabel('lambda')
plt.ylabel('margin')
plt.show
```

Out[ ]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



When choosing the best lambda we should also consider the margin, and the generalization it provides and making sure there is no overfitting despite seeing very clear training and test error results in the first graph. because the norm of w is multiplied by lambda, the higher the lambda the wider the margin, and the lower the lambda the lower are the traning error.

we need to find the balance between the margin and the test errors, between over and under fitting. in this case lambda = 0.05 strikes the best balance out of the bunch, while 0.2 can be argued to be a valid option aswell, we will choose the 0.05 value.

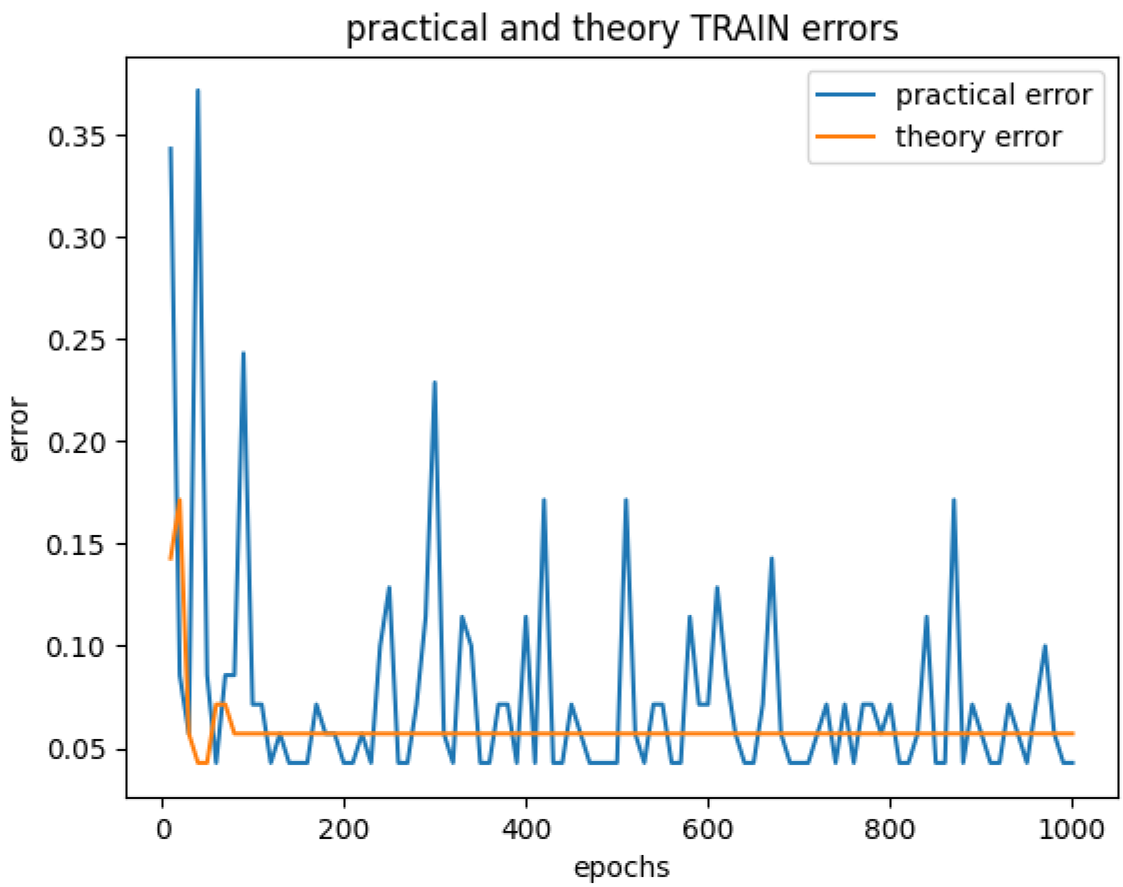
Q7:

In [ ]:

```
ran = range(10,1001,10)
train_err_pr = []
w_train_lst_pr = []
b_train_lst_pr = []
train_err_th = []
w_train_lst_th = []
b_train_lst_th = []
lam = 0.05

for epoch in ran:
  w_train, b_train = svm_with_sgd(X_train, y_train, lam=lam, epochs=epoch, sgd_type= 'practical')
  train_err_pr.append(calculate_error(w_train, b_train, X_train, y_train))
  w_train_lst_pr.append(w_train)
  b_train_lst_pr.append(b_train)

  w_train, b_train = svm_with_sgd(X_train, y_train, lam=lam, epochs=epoch, sgd_type= 'theory')
  train_err_th.append(calculate_error(w_train, b_train, X_train, y_train))
  w_train_lst_th.append(w_train)
```

```
  b_train_lst_th.append(b_train)


plt.plot([i for i in ran], train_err_pr, label= 'practical error')
plt.plot([i for i in ran], train_err_th, label= 'theory error')
plt.legend()
plt.title('practical and theory TRAIN errors')
plt.xlabel('epochs')
plt.ylabel('error')
plt.show
```

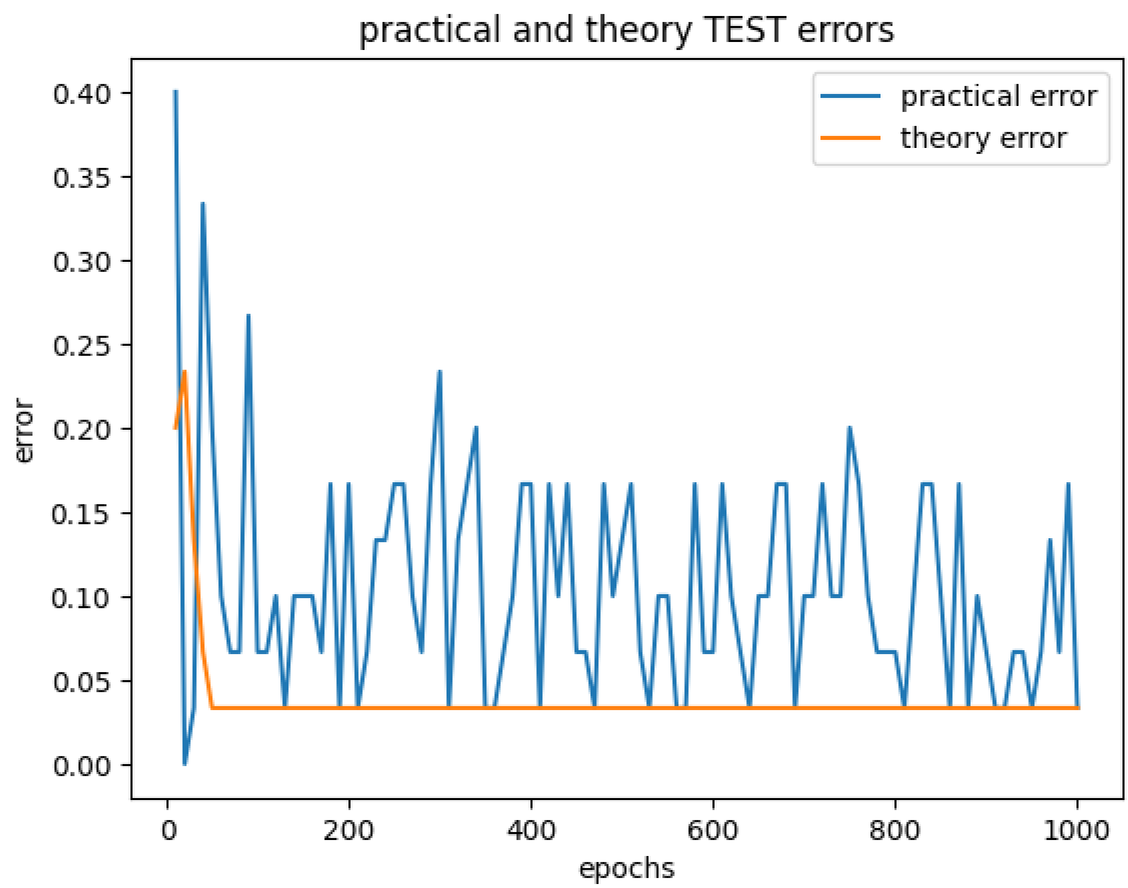
Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



In [ ]:

```
test_err_prac = []
test_err_th = []
i = 0

for epoch in ran:
  test_err_prac.append(calculate_error(w_train_lst_pr[i], b_train_lst_pr[i], X_val, y_val))
  test_err_th.append(calculate_error(w_train_lst_th[i], b_train_lst_th[i], X_val, y_val))
  i += 1

plt.plot([i for i in ran], test_err_prac, label= 'practical error')
plt.plot([i for i in ran], test_err_th, label= 'theory error')

plt.title('practical and theory TEST errors')
plt.legend()
plt.xlabel('epochs')
plt.ylabel('error')
plt.show
```

Out[ ]: <function matplotlib.pyplot.show(close=None, block=None)>



we can see the practical algorithem is much less stable than the theory algorithem, after a few iterations it converges into a solid value just like we saw in the lectures and tutorials. the practical algorithem works and looks point by point while the theoretical one takes adventage of all the w.

Part 3

In [ ]:

```
def cross_validation_error(x, y, model, folds):
  fold_att_lst = np.array_split(x, folds)
  fold_labels_lst = np.array_split(y, folds)
  val_res, train_res = [], []

  for outer in range(folds):
    x_train, y_train = [], []
    for inner in range(folds):
      if inner != outer:
        x_train.extend(fold_att_lst[inner])
        y_train.extend(fold_labels_lst[inner])
    x_train = np.array(x_train)
    y_train = np.array(y_train)
    model.fit(x_train, y_train)

    att_fold = fold_att_lst[outer]
    lab_fold = fold_labels_lst[outer]
    train_res.append(1-(model.score(x_train, y_train)))
    val_res.append(1-(model.score(att_fold, lab_fold)))

  average_train_error = np.mean(train_res)
  average_val_error = np.mean(val_res)
  return (average_train_error, average_val_error)
```

In [ ]:

```python
from sklearn.svm import SVC
def svm_results(X_train, y_train, X_test, y_test):
  lambdas = [10 ** (-4), 10 ** (-2), 1, 10 ** 2, 10 ** 4]
  res_dict = {}
  for lam in lambdas:
    c = 1 / lam
    model = SVC(kernel= 'linear', C=c)
    avrg_train_err, avrg_val_err = cross_validation_error(X_train, y_train, model, 5)
    model.fit(X_train, y_train)
    error = (model.predict(X_test) != y_test).mean()
    res_dict[f'SVM_lambda_{lam}'] = (avrg_train_err, avrg_val_err, error)

  return res_dict
```

In [ ]:

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris_data = load_iris()
X, y = iris_data['data'], iris_data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7)
lambdas = [10 ** (-4), 10 ** (-2), 1, 10 ** 2, 10 ** 4]
average_train_error = []
average_val_error = []
test_error = []
bar = np.arange(5)

dic = svm_results(X_train, y_train, X_test, y_test)
for key in dic.keys():
  average_train_error.append(dic[key][0])
  average_val_error.append(dic[key][1])
  test_error.append(dic[key][2])

plt.bar(bar-0.2, average_train_error, width = 0.2, align='center', label="average train error")
plt.bar(bar, average_val_error, width = 0.2, align='center',  label="average val error")
plt.bar(bar+0.2, test_error, width = 0.2, align='center',  label="test error")

plt.title('average train, validation and test errors')
plt.xlabel('lambda')
plt.ylabel('error')
plt.xticks(bar, lambdas)
plt.legend()
plt.show()
```

we can see that the models both by the cv method and the test error are lambda = 1. as we saw and discussed before, small lambda values will lead to over fitting and big values will lead to underfitting. in this case lambda 1 is had the lowest chance of over and underfitting.

שאלה 2

(1) נרצה להראות כי מחובר אחד סכום פונקציות קמורות, ולכן בעיית ה-soft-SVM היא גם כן פונק' קמורה.

מחובר A: הסכום: $\lambda\|w\|^2$ - ראינו בתרגול כי נורמה היא פונקציה קמורה ולכן קמורה עבור סקלר λ

מחובר B: הסכום $\frac{1}{m}\sum_{i}^{m}\max\{0, 1-y_i(\langle w,x_i\rangle+b)\}$ בגלל ה max שבין 2 פונקציות לינאריות, לכן שתיהם קמורות, מכך נובע כי גם המקסימום שלהם היא פונקציה קמורה ולכן הסכום שלהם הוא קמור כי הוא סכום של פונקציות קמורות, עבור סקלר $\frac{1}{m}$

מכיוון כי מחובר A שני פונקציות קמורות הוא גם כן קמור ולכן ה סכום שלהם הוא גם כן פונקציה קמורה

נסכם והבחנו כי soft-SVM: $(w^*,b^*) = \arg\min \frac{1}{m}\sum_{i}^{m}\max\{0, 1-y_i(\langle w,x_i\rangle+b)\} + \lambda\|w\|^2$ היא פונקציה קמורה.

(2) עבור ההנחה, נחלק למקרים:

(א) $1-y_i\langle w,x_i\rangle < 0$:

$$l(w,x_i,y_i) = 0 \le \|w_2-w_1\|\cdot\max_j\|x_j\|$$

(ב) $1-y_i\langle w,x_i\rangle \ge 0$:

$$|L(w_2,x_i,y_i)-L(w_1,x_i,y_i)| = |y_i\langle w_2,x_i\rangle - y_i\langle w_1,x_i\rangle| \overset{*}{=} |\langle w_2,x_i\rangle - \langle w_1,x_i\rangle| =$$
$$= |\langle w_2-w_1,x_i\rangle| \overset{**}{\le} \|x_i\|\cdot\|w_2-w_1\| \le \max_j\|x_j\|\,\|w_2-w_1\|$$

* $y_i\in\{-1,1\}$

** קושי שוורץ

(ג) שים לב כי ההוכחה עבור המקרה בו $1-y_i\langle w_1,x_i\rangle \ge 0$, $1-y_i\langle w_2,x_i\rangle < 0$ זהה ל מקרה (ב)

ולכן l בנקודה (x_i,y_i) היא R-Lipschitz כפונ' של w כאשר $R:=\max_j\|x_j\|$

(3) סאב גרדיאנט לפי w:

$$g_w^i = \begin{cases} \frac{\partial f}{\partial w}(1-(y_i\langle w,x_i\rangle+b))+\lambda\|w\|^2 = -y_ix_i+2\lambda w & 1-(y_i\langle w,x_i\rangle+b) > 0 \\ \frac{\partial f}{\partial w}\lambda\|w\|^2 = 2\lambda w & 1-(y_i\langle w,x_i\rangle+b) \le 0 \end{cases}$$

סאב גרדיאנט לפי b:

$$g_b^i = \begin{cases} \frac{\partial f}{\partial b}(1-(y_i\langle w,x_i\rangle+b))+\lambda\|w\|^2 = -y_i & 1-(y_i\langle w,x_i\rangle+b) > 0 \\ \frac{\partial f}{\partial b}\lambda\|w\|^2 = 0 & 1-(y_i\langle w,x_i\rangle+b) \le 0 \end{cases}$$

שאלה 4

ג3:

$$\forall u\in R^d : g(u) \geq g(w) + \langle u-w, \nabla g_j(w)\rangle$$

<u>הוכחה</u>: לכל i, g_i קמורה וגזירה ב-R^d $\Leftarrow$ לכל (א״ש הגרדיאנט) u: $g_j(u) \geq g_j(w) + \langle u-w, \nabla g_j(w)\rangle$ (*)

לפי ההגדרה של g: $\forall u, i: g(u) \geq g_i(u)$, ו- $g_j(w)=g(w)$, לכן:

$$\forall u\in R^d: g(u) \geq g_j(u) \overset{(*)}{\geq} g_j(w) + \langle u-w, \nabla g_j(w)\rangle = g_j(w) + \langle u-w, \nabla g_j(w)\rangle$$

$\blacksquare$