Code Listing

Inez Wijnands & Guido Zuidhof
s4149696 & s4160703

exercise1.py

```
# -*- coding: utf-8 -*-
# Inez Wijnands s4149696
# Guido Zuidhof s4160703
# SML ASS 1 exercise1.py

from __future__ import division
import math
import numpy as np
import matplotlib.pyplot as plt


def f(x):
    return 1 + math.sin(8*x + 1)

# A-matrix
def Aij(i,j,x):
    return np.sum(x[n]**(i+j) for n in xrange(len(x)))

# T-vector
def Ti(i,t,x):
    return np.sum(t[n]*(x[n]**i) for n in xrange(len(x)))

#Normally distributed noise
def createNoise(n):
    return np.random.normal(0,0.3,n)

def createData(n=10):
    X = np.linspace(0,1,num = n)
    Y = map(f, X)

    noise = createNoise(n)
    Y+=noise

    D = np.vstack((X,Y))
    return D

def PolCurFit(D,M, _lambda=0):

    x = D[0]
    t = D[1]

    M = M + 1
    #Construct A matrix by calling Aij function for every entry
    A = np.array([[Aij(i,j,x) for j in xrange(M)] for i in xrange(M)])

    #Ridge Regression
    A = A + _lambda * np.identity(len(A))

    #Construct T vector
    T = np.array([Ti(i,t,x) for i in xrange(M)])
    return np.linalg.solve(A,T)

#Polynomial value at x for weight w
def poly_y(x,w):
    y = 0.0

    for i in xrange(len(w)):
        y += w[i]* (x**i)

    return y

def root_mean_square_error(x,t,w):
    return np.sqrt(2*sum_sq_error(x,t,w)/len(x))

def sum_sq_error(x,t,w):
        _sum = 0

        for n in xrange(len(x)):
            _sum +=  (poly_y(x[n],w)-t[n])**2

        _sum = 0.5 * _sum
        return _sum


def run(data_n = 10):
    D = createData(data_n)

    plt.close()
    plt.plot(D[0], D[1], 'r+', label='D'+str(data_n), markersize=10)

    # 1000 points between 0 and 1 (x), for the smooth line
    x_1000 =   np.linspace(0,1,num = 1000)

    plt.plot(x_1000,[f(x) for x in x_1000], 'b',  label='f(x)')

    plt.xlabel('x')
    plt.ylabel('t')

    T = createData(100)

    #Create plot with fitted curves for some values of M
    for M,color in zip([1,3,9],['r','y','g']):
        weights= PolCurFit(D,M)
        plt.plot(x_1000,[poly_y(x,weights) for x in x_1000], color, label='M('+str
(M)+')')

    plt.legend(loc='upper right',ncol=2)
    plt.savefig('1_2_{0}.png'.format(data_n))
    plt.show()

    errors = []
    errorsTest = []
    Ms = range(1,11)

    #Determine errors for various values of M
    for M in Ms:
        weights= PolCurFit(D,M)
        error = root_mean_square_error(D[0],D[1],weights)
        errorTest = root_mean_square_error(T[0],T[1],weights)

        errors.append(error)
        errorsTest.append(errorTest)

    plt.xlabel('$M$')
    plt.ylabel('$E_{RMS}$')

    #Create plot for root mean square errors of D and T
    plt.plot(Ms, errors, label='$\mathcal{D}$')
    plt.plot(Ms, errorsTest, label='$\mathcal{T}$')
    plt.legend(loc='upper left',ncol=2)
    plt.savefig('1_3_{0}.png'.format(data_n))


def run_with_regularization():
    plt.close()
    D = createData(10)
    T = createData(100)

    lambdas = range(-3,16)
    _lambdas = [10**-l for l in lambdas]

    errors = []
    errorsTest = []

    weight7 = []
    weight8 = []
    weight9 = []

    #Run for various values of penalty (lambda)
    for l in _lambdas:
        weights= PolCurFit(D,9,l)

        error = root_mean_square_error(D[0],D[1],weights)
        errorTest = root_mean_square_error(T[0],T[1],weights)
        errors.append(error)
        errorsTest.append(errorTest)

        weight7.append(weights[6])
        weight8.append(weights[7])
        weight9.append(weights[8])


    plt.xlabel('$ln \lambda$')
    plt.ylabel('$E_{RMS}$')

    plt.plot(np.log(_lambdas), errors, label='$\mathcal{D}$')
    plt.plot(np.log(_lambdas), errorsTest, label='$\mathcal{T}$')

    plt.savefig('1_5.png')
    plt.close()


    plt.xlabel('$ln \lambda$')
    plt.ylabel('$Weight value$')

    plt.plot(np.log(_lambdas), weight7, label='$w7$')
    plt.plot(np.log(_lambdas), weight8, label='$w8$')
    plt.plot(np.log(_lambdas), weight9, label='$w9$')
    plt.legend(loc='upper right',ncol=2)
    plt.savefig('1_5_2.png')


if __name__ == "__main__":
    np.random.seed(25)
    run(40)
    run_with_regularization()
```

exercise2.py

```
# -*- coding: utf-8 -*-
# Inez Wijnando s4149696
# Guido Zuidhof s4160703
# SML ASS 1 exercise2.py

from __future__ import division
import numpy as np

import matplotlib.pyplot as plt
from matplotlib import cm


def h(x,y):
    return 100 * ((y-(x**2))**2) + (1-x)**2

#Derivative x
```

```
def h_dx(x,y):
    return - 400 * (x*y) + 400*(x**3) + 2*x - 2

#Derivative y
def h_dy(x,y):
    return 200*y - 200 * (x**2)

# Gradient descent
def sgd(x=-2,y=2, eta=0.001, max_iteration=500000):

    iteration = 0

    x_coords = [x]
    y_coords = [y]

    while iteration < max_iteration :
        step_x = - eta * h_dx(x,y)
        step_y = - eta * h_dy(x,y)

        x = x + step_x
        y = y + step_y

        #print iteration, "X:",x, "Y:",y, "\th(x,y):", h(x,y)
        iteration += 1

        x_coords.append(x)
        y_coords.append(y)


        #Converged? break
        if np.abs(step_x) < eta*(10**-5) and np.abs(step_y) < eta*(10**-5):
            break

    print iteration, "X:",x, "Y:",y, "\th(x,y):", h(x,y)
    return x_coords, y_coords, iteration



if __name__ == "__main__":

    etas = [0.001, 0.0001, 0.00001]

    for eta in etas:

        plt.close()
        fig = plt.figure()

        ax = fig.gca(projection='3d')

        X = np.arange(-2,2,0.03)
        Y = np.arange(-1.0, 3.0, 0.03)
        X, Y = np.meshgrid(X, Y)
```

```
z = np.array( [h(x,y) for x,y in zip(np.ravel(X),np.ravel(Y))])
Z = z.reshape(X.shape)

ax.plot_surface(X, Y, Z,cmap=cm.rainbow)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('H(x,y)')

ax.view_init(elev=90, azim=50)

max_iter = 10000000
x_coords, y_coords, iterations = sgd(eta=eta, max_iteration = max_iter)

converged = 'n' if max_iter == iterations else 'y'

#Plot trajectory
z_coords = [h(x,y) for x,y in zip(x_coords,y_coords)]
ax.plot(x_coords, y_coords, z_coords,  color='#FFFF00')

plt.savefig('trajectory{0}{1}{2}.png'.format(eta,converged,iterations))
plt.show()
```