The laboratory work

Gradient descent

# convex\_grad\_surrogate.py is a toy wrapper to illustrate the path

# taken by gradient descent. The steps are evaluated

# at the objective, and then plotted. For the first 5 iterations the

# linear surrogate used to transition from point to point is also plotted.

# The plotted points on the objective turn from green to red as the

# algorithm converges (or reaches a maximum iteration count, preset to 50).

#

# The (convex) function here is

#

# g(w) = log(1 + exp(w^2))

from numpy import \*

from matplotlib.pyplot import \*

#from pylab import \*

import time

def obj(y):

 z = log(1 + exp(y\*\*2))

 return z

def grad(y):

 z = (2\*exp(y\*\*2)\*y)/(exp(y\*\*2) + 1)

 return z

def surrogate(y,x):

 z = obj(y) + grad(y)\*(x - y)

 return z

###### ML Algorithm functions ######

def gradient\_descent(w0,alpha):

 w = w0

 obj\_path = []

 w\_path = []

 w\_path.append(w0)

 obj\_path.append(log(1 + exp(w\*\*2)))

 # start gradient descent loop

 grad = 1

 iter = 1

 max\_its = 50

 while linalg.norm(grad) > 10\*\*(-5) and iter <= max\_its:

 # take gradient step

 grad = (2\*exp(w\*\*2)\*w)/(exp(w\*\*2) + 1)

 w = w - alpha\*grad

 # update path containers

 w\_path.append(w)

 obj\_path.append(log(1 + exp(w\*\*2)))

 iter+= 1

 # show final average gradient norm for sanity check

 s = grad\*\*2

 s = 'The final average norm of the gradient = ' + str(float(s))

 print(s)

 # # for use in testing if algorithm minimizing/converging properly

 # obj\_path = asarray(obj\_path)

 # obj\_path.shape = (iter,1)

 # plot(asarray(obj\_path))

 # show()

 return (w\_path,obj\_path)

###### plotting functions #######

def main():

 alpha = 0.12

 x = [1, 3, 5, 10, 15, 20]

 #scatter (x,obj(x), s=420, c='g', alpha=1)

 w0 = float(x[0])

 w\_path,obj\_path = gradient\_descent(w0,alpha) # perform gradient descent

main()