LAB2 OPTIMIZATION FOR MACHINE LEARNING

FALL SEMESTER 2023

Kshitij Jaiswal

B20CS028

Problems Solved - 2, 3, 4, 14

**Code for Problem 2 -**

#kshitij jaiswal

#b20cs028

import numpy as np

from cvxopt import solvers, matrix

r = 8

nVal = [5,10,30]

pi = np.pi

lb=-1.0

ub=1.0

def I(n):

I1 = []

for i in range(2, n+1):

if (i&1) > 0:

I1.append(i)

I2 = []

for i in range(2, n+1):

if (i&1) == 0:

I2.append(i)

return (I1, I2)

def fun1(params, n):

I1, I2 = I(n)

x1 = params[0]

t = 0.0

for i in I1:

t += ((params[i-1] - np.sin(6\*pi\*x1 + i\*pi/n))\*\*2)

t \*= (2/np.linalg.norm(I1))

t += x1

return t

def fun2(params, n):

I1, I2 = I(n)

x1 = params[0]

t = 0.0

for i in I2:

t += ((params[i-1] - np.sin(6\*pi\*x1 + i\*pi/n))\*\*2)

t \*= (2/np.linalg.norm(I2))

t -= np.sqrt(x1)

t += 1

return t

def gradNhessFun1(params):

#x\_i = x0

g=[]

numParams = len(params)

F = fun1(params, numParams)

h, g = 1e-5, np.zeros(numParams)

fun = []

for i in range(numParams):

params[i] += h

f = fun1(params, numParams)

g[i] = (f - F)/h

fun.append(f)

params[i] -= h

H = np.matrix(np.zeros((numParams, numParams)))

for i in range(numParams):

params[i] += h

for j in range(i+1):

params[j] += h

H[i,j] = (fun1(params, numParams) - fun[i] - fun[j] + F)/(h\*\*2)

H[j,i] = H[i,j]

params[j] -= h

params[i] -= h

return (g,H)

def gradNhessFun2(params):

g=[]

numParams = len(params)

F = fun2(params, numParams)

h, g = 1e-5, np.zeros(numParams)

fun = []

for i in range(numParams):

params[i] += h

f = fun2(params, numParams)

g[i] = (f - F)/h

fun.append(f)

params[i] -= h

H = np.matrix(np.zeros((numParams, numParams)))

for i in range(numParams):

params[i] += h

for j in range(i+1):

params[j] += h

H[i,j] = (fun2(params, numParams) - fun[i] - fun[j] + F)/(h\*\*2)

H[j,i] = H[i,j]

params[j] -= h

params[i] -= h

return (g,H)

if \_\_name\_\_ == '\_\_main\_\_':

for n in nVal:

params = []

for i in range(n):

if i==0:

params.append(np.random.uniform(0.001, ub))

else:

params.append(np.random.uniform(lb, ub))

g1, H1 = gradNhessFun1(params)

g2, H2 = gradNhessFun2(params)

g = [g1, g2]

H = [H1, H2]

params = np.array(params)

for i in range(2):

lmbda = min(np.linalg.eig(H[i])[0])

if lmbda > 0.01:

lmbda = 0

else:

lmbda = 0.01-lmbda

for j in range(n):

H[i][j,j] += lmbda

A = np.vstack((-np.identity(n), np.identity(n)))

b = np.append(params-lb, ub-params)

b[n] = params[0] - 0.001

sol = solvers.qp(matrix(H[i], tc = 'd'), matrix(g[i]), matrix(A), matrix(b))

print("Optimal soln for number of parameters " + str(n) + " and f" + str(i+1) + " -- \n", np.array(sol['x']))

print('\n')

**Code for Problem 3 -**

#kshitij jaiswal

#b20cs028

import numpy as np

from cvxopt import solvers, matrix

r = 8

nVal = [5,10,30]

pi = np.pi

lb=-1.0

ub=1.0

def I(n):

I1 = []

for i in range(2, n+1):

if (i&1) > 0:

I1.append(i)

I2 = []

for i in range(2, n+1):

if (i&1) == 0:

I2.append(i)

return (I1, I2)

def fun1(params, n):

I1, I2 = I(n)

x1 = params[0]

t = 0.0

for i in I1:

t1 = 0.3\*(x1\*\*2)\*np.cos(24\*pi\*x1 + (4\*i\*pi)/n) + 0.6\*x1

t2 = np.cos(6\*pi\*x1 + (i\*pi)/n)

t += ((params[i-1] - t1\*t2)\*\*2)

t \*= (2/np.linalg.norm(I1))

t += x1

return t

def fun2(params, n):

I1, I2 = I(n)

x1 = params[0]

t = 0.0

for i in I2:

t1 = 0.3\*(x1\*\*2)\*np.cos(24\*pi\*x1 + (4\*i\*pi)/n) + 0.6\*x1

t2 = np.sin(6\*pi\*x1 + (i\*pi)/n)

t += ((params[i-1] - t1\*t2)\*\*2)

t \*= (2/np.linalg.norm(I2))

t -= np.sqrt(x1)

t += 1

return t

def gradNhessFun1(params):

g=[]

numParams = len(params)

F = fun1(params, numParams)

h, g = 1e-5, np.zeros(numParams)

fun = []

for i in range(numParams):

params[i] += h

f = fun1(params, numParams)

g[i] = (f - F)/h

fun.append(f)

params[i] -= h

H = np.matrix(np.zeros((numParams, numParams)))

for i in range(numParams):

params[i] += h

for j in range(i+1):

params[j] += h

H[i,j] = (fun1(params, numParams) - fun[i] - fun[j] + F)/(h\*\*2)

H[j,i] = H[i,j]

params[j] -= h

params[i] -= h

return (g,H)

def gradNhessFun2(params):

g=[]

numParams = len(params)

F = fun2(params, numParams)

h, g = 1e-5, np.zeros(numParams)

fun = []

for i in range(numParams):

params[i] += h

f = fun2(params, numParams)

g[i] = (f - F)/h

fun.append(f)

params[i] -= h

H = np.matrix(np.zeros((numParams, numParams)))

for i in range(numParams):

params[i] += h

for j in range(i+1):

params[j] += h

H[i,j] = (fun2(params, numParams) - fun[i] - fun[j] + F)/(h\*\*2)

H[j,i] = H[i,j]

params[j] -= h

params[i] -= h

return (g,H)

if \_\_name\_\_ == '\_\_main\_\_':

grad = []

hess = []

for n in nVal:

params = []

for i in range(n):

if i==0:

params.append(np.random.uniform(0.001, ub))

else:

params.append(np.random.uniform(lb, ub))

g1, H1 = gradNhessFun1(params)

g2, H2 = gradNhessFun2(params)

g = [g1, g2]

H = [H1, H2]

params = np.array(params)

for i in range(2):

lmbda = min(np.linalg.eig(H[i])[0])

if lmbda > 0.01:

lmbda = 0

else:

lmbda = 0.01-lmbda

for j in range(n):

H[i][j,j] += lmbda

A = np.vstack((-np.identity(n), np.identity(n)))

b = np.append(params-lb, ub-params)

b[n] = params[0] - 0.001

sol = solvers.qp(matrix(H[i], tc = 'd'), matrix(g[i]), matrix(A), matrix(b))

print("Optimal soln for number of parameters " + str(n) + " and f" + str(i+1) + " -- \n", np.array(sol['x']))

print('\n')

**Code for Problem 4 -**

#kshitij jaiswal

#b20cs028

import numpy as np

from cvxopt import solvers, matrix

r = 8

nVal = [5,10,30]

pi = np.pi

lb=-1.0

ub=1.0

def I(n):

I1 = []

for i in range(2, n+1):

if (i&1) > 0:

I1.append(i)

I2 = []

for i in range(2, n+1):

if (i&1) == 0:

I2.append(i)

return (I1, I2)

def fun1(params, n):

I1, I2 = I(n)

x1 = params[0]

t = 0.0

summ = 0.0

prod = 1.0

for i in I1:

yi = params[i-1] - np.power(x1, 0.5 + 1.5\*(i-2)/(n-2))

summ += (yi\*\*2)

prod \*= np.cos((20\*yi\*pi)/np.sqrt(i))

t = 4\*summ - 2\*prod + 2

t \*= (2/np.linalg.norm(I1))

t += x1

return t

def fun2(params, n):

I1, I2 = I(n)

x1 = params[0]

t = 0.0

summ = 0.0

prod = 1.0

for i in I2:

yi = params[i-1] - np.power(x1, 0.5 + 1.5\*(i-2)/(n-2))

summ += (yi\*\*2)

prod \*= np.cos((20\*yi\*pi)/np.sqrt(i))

t = 4\*summ - 2\*prod + 2

t \*= (2/np.linalg.norm(I2))

t -= np.sqrt(x1)

t += 1

return t

def gradNhessFun1(params):

g=[]

numParams = len(params)

F = fun1(params, numParams)

h, g = 1e-5, np.zeros(numParams)

fun = []

for i in range(numParams):

params[i] += h

f = fun1(params, numParams)

g[i] = (f - F)/h

fun.append(f)

params[i] -= h

H = np.matrix(np.zeros((numParams, numParams)))

for i in range(numParams):

params[i] += h

for j in range(i+1):

params[j] += h

H[i,j] = (fun1(params, numParams) - fun[i] - fun[j] + F)/(h\*\*2)

H[j,i] = H[i,j]

params[j] -= h

params[i] -= h

return (g,H)

def gradNhessFun2(params):

g=[]

numParams = len(params)

F = fun2(params, numParams)

h, g = 1e-5, np.zeros(numParams)

fun = []

for i in range(numParams):

params[i] += h

f = fun2(params, numParams)

g[i] = (f - F)/h

fun.append(f)

params[i] -= h

H = np.matrix(np.zeros((numParams, numParams)))

for i in range(numParams):

params[i] += h

for j in range(i+1):

params[j] += h

H[i,j] = (fun2(params, numParams) - fun[i] - fun[j] + F)/(h\*\*2)

H[j,i] = H[i,j]

params[j] -= h

params[i] -= h

return (g,H)

if \_\_name\_\_ == '\_\_main\_\_':

grad = []

hess = []

for n in nVal:

params = []

for i in range(n):

if i==0:

params.append(np.random.uniform(0.001, ub))

else:

params.append(np.random.uniform(lb, ub))

g1, H1 = gradNhessFun1(params)

g2, H2 = gradNhessFun2(params)

g = [g1, g2]

H = [H1, H2]

params = np.array(params)

for i in range(2):

lmbda = min(np.linalg.eig(H[i])[0])

if lmbda > 0.01:

lmbda = 0

else:

lmbda = 0.01-lmbda

for j in range(n):

H[i][j,j] += lmbda

A = np.vstack((-np.identity(n), np.identity(n)))

b = np.append(params-lb, ub-params)

b[n] = params[0] - 0.001

sol = solvers.qp(matrix(H[i], tc = 'd'), matrix(g[i]), matrix(A), matrix(b))

print("Optimal soln for number of parameters " + str(n) + " and f" + str(i+1) + " -- \n", np.array(sol['x']))

print('\n')

**Code for Problem 14 -**

#kshitij jaiswal

#b20cs028

import numpy as np

from cvxopt import solvers, matrix

r = 8

pair = [(2,2), (3,3), (3,10)]

pi = np.pi

lb=0.0

ub=1.0

def fun(params, n, m):

f = []

gx = 0.0

for i in range(m, n+1):

gx += ((params[i-1]-0.5)\*\*2)

for i in range(m):

prod = 1.0

if i==0:

for j in range (1, m):

prod \*= (np.cos(0.5\*pi\*params[j-1]))

else:

for j in range (1, m-i+1):

prod \*= ((np.cos(0.5\*pi\*params[j-1]))\*(np.sin(0.5\*pi\*params[m-j])))

prod \*= (1+gx)

f.append(prod)

return f

def gradNhessFun(params, m):

#x\_i = x0

grad=[]

hess=[]

numParams = len(params)

h = 1e-5

F = fun(params, numParams, m)

funGrad1 = []

funGrad2 = []

for i in range(numParams):

params[i] += h

f = fun(params, numParams, m)

funGrad1.append(f)

params[i] -= h

for i in range(numParams):

params[i] += h

aux = []

for j in range(numParams):

params[j] += h

f = fun(params, numParams, m)

aux.append(f)

params[j] -= h

funGrad2.append(aux)

params[i] -= h

for i in range(m):

g = np.zeros(numParams)

for j in range(numParams):

g[j] = (funGrad1[j][i] - F[i])/h

H = np.matrix(np.zeros((numParams, numParams)))

for j in range(numParams):

for k in range(i+1):

H[j,k] = (funGrad2[j][k][i] - funGrad1[j][i] - funGrad1[k][i] + F[i])/(h\*\*2)

H[k,j] = H[j,k]

grad.append(g)

hess.append(H)

return (grad,hess)

if \_\_name\_\_ == '\_\_main\_\_':

grad = []

hess = []

for (m,n) in pair:

params = []

for i in range(n):

params.append(np.random.uniform(lb, ub))

grad, hess = gradNhessFun(params, m)

params = np.array(params)

A = np.vstack((-np.identity(n), np.identity(n)))

b = np.append(params-lb, ub-params)

for i in range(m):

g = grad[i]

H = hess[i]

lmbda = min(np.linalg.eig(H)[0])

if lmbda > 0.01:

lmbda = 0

else:

lmbda = 0.01-lmbda

for j in range(n):

H[j,j] += lmbda

sol = solvers.qp(matrix(H, tc = 'd'), matrix(g), matrix(A), matrix(b))

print("Optimal soln for number of parameters " + str(n) + " and f" + str(i+1) + " -- \n", np.array(sol['x']))

print('\n')