Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
 высшего образования   
«Национальный исследовательский университет «МЭИ»

Кафедра «Релейная защита и автоматизация энергосистем»

Лабораторная работа №1

«**ИССЛЕДОВАНИЕ ИТЕРАТИВНЫХ МЕТОДОВ ЧИСЛЕННОЙ  
ОПТИМИЗЦИИ**»

Выполнил: Максимов Р.С.

Группа: Э-13м-19

Проверил: Болтунов А.П.

Москва 2020

**Цель работы:** Ознакомление с методами поиска экстремума нелинейной выпуклой функции нескольких переменных методом градиентного спуска и методом имитации отжига.

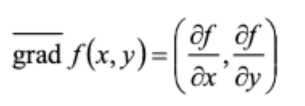
Описание методов.

1. Метод градиентного спуска.

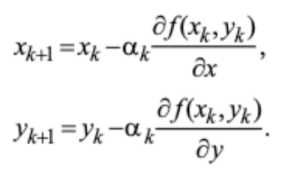
Задача состоит в отыскании минимума функции двух переменных *f*(*x*,*y*) (следует отметить, что если необходимо найти максимум некоторой функции *F*(*x*,*y*), то эта задача сводится к поиску минимума функции *f*(*x*,*y*)=-*F*(*x*,*y*) ).

Большинство численных методов состоит в отыскании некоторой последовательности *(xo,yo),* (*x*i,*y*i),..,(*x*k,*yk*), которая при *к®^* (или при *к®кМ)* сходится к точке минимума (*x*\*,*y*\*). Если при этом выполняется *f*(*x0*,*y*0)>*f*(*x1*,*y*i)>..>*f*(*x*k,*y*k), то есть значения функции монотонно убывают при увеличении *к*, то такой метод называется методом спуска.

Известно, что вектор градиента функции:

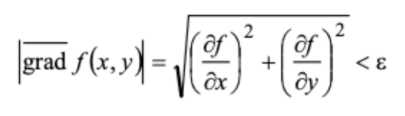


направлен в сторону наибольшего возрастания функции *f*(*x,y*). Поэтому в качестве направления движения можно принять противоположное градиенту направление (антиградиент), т.е. координаты точек пересчитываются по формулам



Выбор величины *ak,* с которой связана длина *к*-го шага, в общем случае является сложной задачей. Если ak мало, то движение будет слишком медленным и потребует значительного объема вычислений. Если a*k* велико, то существует возможность перескочить точку минимума и выйти на противоположный склон функции. При этом возможно нарушение требования монотонного убывания последовательности *f*(*x*k,*yk*) и появляется опасность зацикливания, то есть колебания последовательности (*xk*,*yk*) в некоторой окрестности точки минимума (*x*\*,*y*\*) без приближения к ней. Именно поэтому в алгоритме использовалось два списка для коэф. Альфы: один определяет альфу в зависимости от величины градиента, второй предназначен для выхода из локального минимума и поиска нового значения

Критерием окончания счета принимается неравенство:



либо одновременное выполнение двух неравенств, либо окончания времени расчета.

def gradient\_low(f,diapasonx,diapazony, delta):

ix = []

iy = []

it = []

start\_time\_s = time.time()

end\_time\_s = 0

deltaX = delta

deltaY = delta

eps = delta

t = True

fullGrad = 999999

gradX = 999999

gradY = 9999999

i = 0

lastx = getStartValue(diapasonx)

lasty = getStartValue(diapazony)

x = lastx

y = lasty

ix.append(x)

iy.append(y)

it.append(i)

minimial = [ 999 , "No", "No"] #f,x,y

# print("\n"+"Start X = ",x)

# print("Start Y = ",y)

number = 0

while ((fullGrad > eps or (np.abs(gradX) > deltaX/2 and np.abs(gradY) > deltaY/2)) and end\_time\_s < 1.):

# print("\n"+"Start X = ",x)

# print("Start Y = ",y))

function = f(float(x), float(y))

if (minimial[0] > function):

minimial[0] = function

minimial[1] = x

minimial[2] = y

randomIndex = int(np.random.randint(0, 10, 1))

alfa = alfrandom[randomIndex]

# print("Value of Function = ",function)

fullnextFunction = f(float(x-alfa), float(y-alfa))

# print("Next Value of Function = ",fullnextFunction)

nextFunctionX = f(float(x-alfa), y)

nextFunctionY = f(x, float(y-alfa))

gradX = (nextFunctionX - function)/(-alfa)

orderX = order(gradX)

if (orderX > len(al)-1):

alfaX = al[len(al)-1]\*int(np.random.randint(0, 10, 1))

else:

alfaX = al[orderX]\*int(np.random.randint(0, 10, 1))

gradY = (nextFunctionY - function)/(-alfa)

orderY = order(gradY)

if (orderY > len(al)-1):

alfaY = al[len(al)-1]\*int(np.random.randint(0, 10, 1))

else:

alfaY = al[orderY]\*int(np.random.randint(0, 10, 1))

# print("Gradient X = ",gradX)

# print("Gradient Y = ",gradY)

fullGrad = np.sqrt(np.power(gradX,2) + np.power(gradY,2))

x = float(x - alfaX\*np.power(gradX,2)/gradX)

y = float(y - alfaY\*np.power(gradY,2)/gradY)

if (np.abs(fullnextFunction - function) < eps):

number = number +1

if (number > 35):

x = getStartValue(diapasonx)

y = getStartValue(diapazony)

x = checkDiapason(x,diapasonx,lastx)

y = checkDiapason(y,diapazony,lasty)

number = 0

else:

x = float(x - (getRandomValue(x,alfrandom)\*np.power(gradX,2)/gradX)/0.4)

y = float(y - (getRandomValue(y,alfrandom)\*np.power(gradY,2)/gradY)/0.4)

x = checkDiapason(x,diapasonx,lastx)

y = checkDiapason(y,diapazony,lasty)

lastx = x

lasy = y

ix.append(x)

iy.append(y)

i = i + 1

it.append(i)

end\_time\_s = float(time.time() - start\_time\_s)

print("Min value of function = ",minimial[0])

print("Min value of X = ",minimial[1])

print("Min value of Y = ",minimial[2])

lines = [ix,iy]

lines[0],lines[1] = plt.plot(it, ix, "r",it, iy, "b")

plt.legend(lines,['X','Y'],loc='best')

plt.ylabel('y,x')

plt.xlabel("i шаг")

plt.grid()

plt.show()

# plt.show()

return minimial[0],minimial[1],minimial[2]

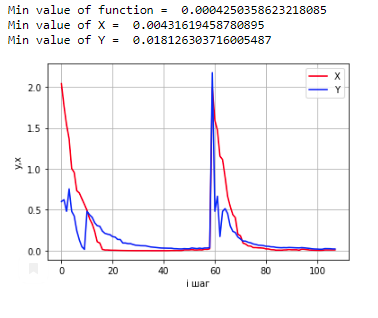


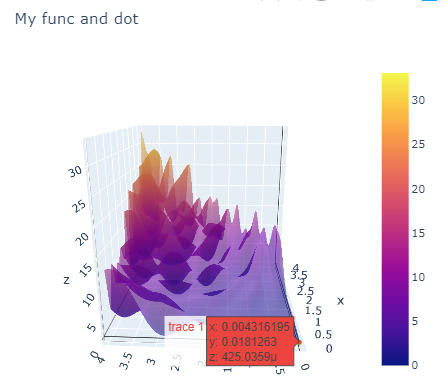
Figure 1 - Поиск оптимальных значений x,y градиентным спуском

В функцию gradient\_low(f,diapasonx,diapazony,delta) передается исследуемая функция, диапазон для x, диапазон для y и точность для вышеуказанных условий. Пример:

grad = gradient\_low(XYZcreator,[0,4],[0,4], 0.01)

Функция gradient\_low возвращает минимальное значение исследуемой функции, а также x,y. Посмотрим граф для исследуемой функции с найденными минимальными точками:

graph(XYZcreator,grad[1],grad[2])



1. Имитация отжига

Применение данной схемы к оптимизации основано на том, что локальное (субоптимальное) решение, найденное в процессе решения задачи оптимизации, также можно рассматривать как дефектное решение. Улучшить это решение (приблизиться к глобальному оптимуму) можно путём его случайных флюктуаций, амплитуда которых уменьшается с ростом номера итераций. Принципиальным в алгоритме SA является то, что, в отличие от большинства других стохастических алгоритмов поисковой оптимизации, он допускает шаги, приводящие к увеличению значений фитнес-функции. Алгоритм SA относится к классу так называемых пороговых стохастических алгоритмов безусловной оптимизации.

Пороговый алгоритм в процессе поиска допускает ухудшение значений фитнес-функции до заданного порога, и этот порог в процессе итераций последовательно снижается до нуля. В алгоритме SA величина представляет собой случайную величину с математическим ожиданием, равным *T*, которому придается смысл «температуры» отжигаемого металла. Таким образом, в алгоритме SA переход от решения к решению допускается с вероятностью 0.8:



Алгоритм:

1. На входе минимальная температура , начальная температура 
2. Задаём произвольное первое состояние 
3. 
4. Пока 
   1. 
   2. 
   3. Если , тогда 
   4. Если  переход осуществляется с вероятностью:
   5. Понижаем температуру
5. Возвращаем последнее состояние s

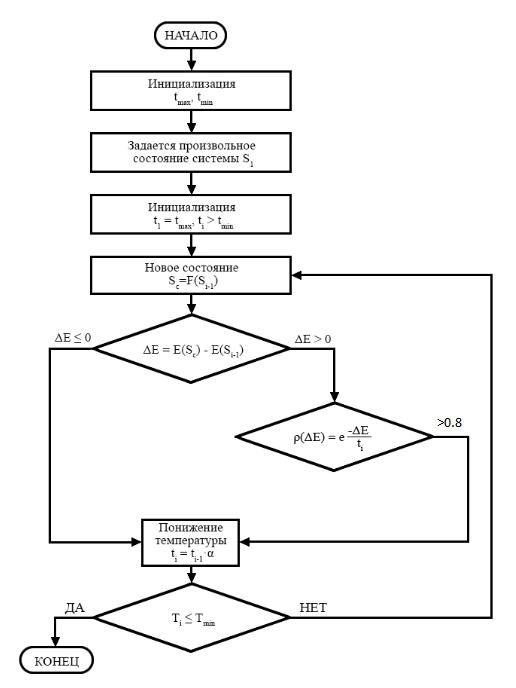


Figure 2 - Алгоритм для имитация отжига

Для оптимального нахождения для каждого параметра был применен такой подходя для решения задачи оптимизации. Новое состояние системы создается на основе рандомного изменения x,y, но в пределах входного диапозона.

Код программы:

def simulated\_annealing(start\_En,start\_T,min\_T,diapasonx,diapazony):

startEnergy = float(start\_En)

startTemperature = float(start\_T)

minTemperaturex = float(min\_T)

minTemperaturey = float(min\_T)

okP = 0.8 #acceptable probability

diapasonX = diapasonx

diapasonY = diapazony

Temperaturex = startTemperature

Temperaturey = startTemperature

Energy = startEnergy

# rang = 0

# coef = 20

start\_time\_s = time.time()

end\_time\_s = 0

x = getStartValue(diapasonX)

y = getStartValue(diapasonY)

lastx = x\*0.88

lasty = y\*0.88

bestStatus = [9999, x,y]

Status = [9999, x,y]

#20 ms for searching

print("\n"+"Start X = ",x)

print("Start Y = ",y)

""" чем больше время, тем больше точность """

while (Temperaturex > minTemperaturex and Temperaturey > minTemperaturey and end\_time\_s < 1.):

lasty = y

y = y - getRandomValue(y,alfrandom)\*int(np.random.randint(-1, 2, 1))/0.1

y = checkDiapason(y,diapasonY,lasty)

lastx = x

x = x - getRandomValue(y,alfrandom)\*int(np.random.randint(-1, 2, 1))/0.1

x = checkDiapason(x,diapasonX,lastx)

Energy = function(x,y)\*200

nextEnergyx = function(x,lasty)\*200

nextEnergyy = function(lastx,y)\*200

deltaEx = nextEnergyx - Energy

deltaEy = nextEnergyy - Energy

# print("\nnextEnergyx = ",nextEnergyx)

# print("nextEnergyy = ",nextEnergyy)

# print("energy = ",Energy)

# print("delta Ex = ",deltaEx)

# print("delta Ey = ",deltaEx)

# print("X1 = ",x)

# print("Y1 = ",y)

# print("last x = ",lastx)

# print("last y = ",lasty)

# print("Tx = ",Temperaturex)

# print("Ty = ",Temperaturey)

if (bestStatus[0] > Status[0]):

# print(bestStatus[0])

bestStatus[0] = Status[0]

bestStatus[1] = Status[1]

bestStatus[2] = Status[2]

if (deltaEx >= 0):

P = getProbability(deltaEx,Temperaturex)

# print("P x = ",P)

if (P > okP):

Status[0] = function(x,y)

Status[1] = x

Status[2] = y

Temperaturex = Temperaturex\*0.999

else:

Status[0] = function(x,y)

Status[1] = x

Status[2] = y

Temperaturex = Temperaturex\*0.999999

if (deltaEy >= 0):

P = getProbability(deltaEy,Temperaturey)

# print("P y = ",P)

if (P > okP):

Status[0] = function(x,y)

Status[1] = x

Status[2] = y

Temperaturey = Temperaturey\*0.999

else:

Status[0] = function(x,y)

Status[1] = x

Status[2] = y

Temperaturey = Temperaturey\*0.999999

end\_time\_s = float(time.time() - start\_time\_s)

print("Time, c = ", end\_time\_s)

return bestStatus

Вывод программы:

Start X = 0.672

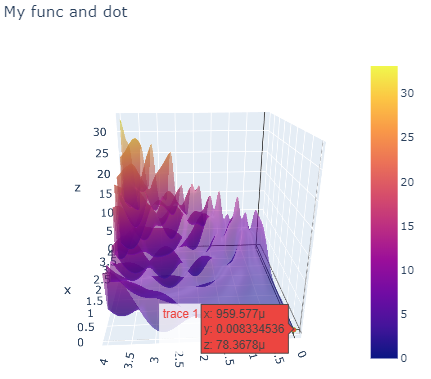
Start Y = 3.968

Time, c = 0.7200474739074707

[7.83677964546093e-05, 0.0009595769611084514, 0.00833453601738117]

Скорость сходимости алгоритма SA в значительной мере определяет вид функции уменьшения температуры. В данной реализации использовался медленное изменение температуры по следующему закону: X= X\*0.9999

Посмотрим граф для исследуемой функции с найденными минимальными точками:



Вывод: данные реализации алгоритмов можно использовать для нахождения не только минимумов функции, но и поиска максимума. При сравнении этих двух алгоритмов было выявлено:

* Имитация отжига возвращает наиболее оптимальные параметры по сравнению с градиентным спуском
* Оба алгоритма подходят для нахождения как минимумов, так и максимумов
* Градиентный спуск может все равно упасть в локальный минимум, для этого необходимо увеличить расчетное время.