پروژه ژنتیک:

هدف این پروژه پیدا کردن تابع اولیه با استفاده از الگوریتم ژنتیک است.

در ابتدای کار ما باید expression tree های رندومی تولید کنیم که برای این کار کلاس درخت را میسازیم.

```
class exp_tree:
    def __init__(self, postfix_exp): #[21x7*+*]
        self.exp = postfix_exp
        self.root = None
        self.parent = None
        self.createTree(self.exp)
```

برای ورودی درخت یعنی postfix_exp تابعی مینویسیم که به صورت ارایه اعداد و عملگرها را در کنار هم به صورتی که valid باشد قرار میدهد.

```
def exp_tree_input(n):
    trees = []
    treezzz = []
    for i in range(n):
        operators_num = np.random.randint(2,15)
        if(operators_num==0):
            trees.append([np.random.randint(1,10)])
        else:
            counter = 1
            length = operators_num*2 + 1
            tree = []
            tree.append(np.random.randint(1,10))
            tree.append(np.random.randint(1,10))
            while(len(tree)!= length and ((length-len(tree))>counter)):
                to_select = np.random.randint(3)
                if(counter <=0 or to_select!=2):
                    if(to_select==0):
                        tree.append(np.random.randint(1,10))
                        counter += 1
                    elif(to_select==1):
                        tree.append('x')
                        counter += 1
                    else:
                        to_select = np.random.randint(2)
                        if(to_select==0):
                            tree.append(np.random.randint(1,10))
                            counter += 1
                        elif(to_select==1):
                            tree.append('x')
                            counter += 1
                else:
                    tree.append(np.random.choice(optr))
                    counter -= 1
            if not(length-len(tree)>counter):
                for i in range(counter):
                    tree.append(np.random.choice(optr))
            trees.append([tree,Valid_Function(tree)])
            et = exp_tree(tree)
            treezzz.append([et,Valid_Function(tree)])
        Generations.append(treezzz)
    return (trees)
```

برای مثال تعداد اعداد و ایکس باید یکی بیشتر از تعداد عملگرها باشند و یا حداقل 2 عضو اول ارایه باید عدد باشند و همچنین عضو آخر آرایه به دلیل اینکه در root قرار می گیرد باید عملگر باشد.

تابع valid_Function به این منظور نوشته شده که آرایه ساخته شده در exp_tree_input را میگیرد و به ازای متغیر اعداد مختلفی را میگذارد و محاسبه میکند و در اخر میانگین خطا را برمیگرداند.

```
def Valid_Function(tree):
    errors = []
    for i in range(len(inputs)):
        mystack = []
        for x in tree:
            if(x not in optr):
                mystack.append(x)
            else:
                tmp1 = mystack.pop()
                tmp2 = mystack.pop()
                if(tmp1=='x'):
                    tmp1 = inputs[i]
                if(tmp2=='x'):
                    tmp2 = inputs[i]
                if(x == '+'):
                    mystack.append(tmp1+tmp2)
                if(x == '-'):
                    mystack.append(tmp1-tmp2)
                if(x == '^'):
```

در قسمت بعد select می کنیم.به معنی که انتخاب میکنیم چه مقدار از درخت های نسل قبل به نسل بعدی انتقال پیدا کنند.
که برای بهتر شدن کار،درخت ها را براساس خطایشان sort میکنیم و به اندازه درصد مورد نظر درخت ها را برای نسل بعد انتخاب می کنیم.

```
def Generic_Select(n):
    if(len(Generations)>1):
        Last_Gen = Generations[-1]
        Last_Gen.sort(key=lambda x : x[1])
        Next_Gen = Last_Gen[:int(0.8*n)]
        Previous_Gen = Generations[-2]
        Previous_Gen.sort(key=lambda x : x[1])
        Next_Gen += Previous_Gen[:int(0.2*n)]
        Generations.append(Next_Gen)
    else:
        Last_Gen = Generations[-1]
        Last_Gen.sort(key=lambda x : x[1])
        Next_Gen = Last_Gen[:n]
        Generations.append(Next_Gen)
```

در قسمت بعدی mutation را داریم.در این قسمت به صورت رندوم تعدادی درخت در نسل اخر را انتخاب میکنیم.سپس به تعداد رندوم در درخت پیمایش میکنیم و یک عدد یا عملگر را تغییر میدهیم،این هم به صورت رندوم.

```
def mutation():
    countTree2mutation = [int(x) for x in np.random.uniform(0,len(Generations[-1]),int(0.5*len(Generations[-1])))]
    for indtree in countTree2mutation:
        count2muation = np.random.randint(0,len(Generations[-1][indtree][0]))
        for i in range(count2muation):
            which2mutate = np.random.randint(1,math.log2(len(Generations[-1][0]))+1)
            tomutate = Generations[-1][indtree][0].root
            while(which2mutate>0):
                toselect = np.random.randint(2)
                if (toselect==0):
                    tomutate = tomutate.right
                    which2mutate -= 1
                else:
                    tomutate = tomutate.left
                    which2mutate -= 1

            if (tomutate.data in optr):
                tomutate.data = np.random.choice(optr)
            else:
                XorInt = np.random.randint(3)
                if (XorInt==1 or XorInt==1):
                    tomutate.data = np.random.randint(1,10)
                else:
                    tomutate.data = 'x'
    temp = Generations[-1][indtree][0]
    del Generations[-1][indtree]
    Generations[-1].append([temp,Valid_Function(temp.exp)])
```

اینکه چه اندازه طول درخت پایین بیاییم و راست یا چپ را انتخاب کنیم به صورت رندوم است.در آخر درخت جدید را با میزان خطایش به نسل آخر درخت ها اضافه میکنیم.

در مرحله بعدی crossover میکنیم.یعنی دو node از2 درخت را به دلخواه و رندوم انتخاب میکنیم و جای این 2 node را با یکدیگر جابه جا میکنیم و یا یک درخت یا یک قسمت از درخت را به درخت دیگر اضافه میکنیم و دوباره درخت جدید را با میزان خطایش در نسل اخر اضافه میکنیم.

در قسمت اخر crossover نسل آخر درخت ها را بر اساس میزان خطای آن ها sort میکنیم و اولین درخت را به عنوان نتیجه برمیگردانیم.

```
def Crossover():
    tree2crossover = [int(x) for x in np.random.uniform(0,len(Generations[-1]),int(0.8*len(Generations[-1])))]
    while(len(tree2crossover)>1):
        indOftree1 = np.random.choice(tree2crossover)
        tree2crossover.remove(indOftree1)
        indOftree2 = np.random.choice(tree2crossover)
        tree2crossover.remove(indOftree2)

        rangetree1 = np.random.randint(0,math.log2(len(Generations[-1][indOftree1][0])))
        rangetree2 = np.random.randint(0,math.log2(len(Generations[-1][indOftree2][0])))

        node2crossover1 = Generations[-1][indOftree1][0].root

        while(rangetree1>0 ):
            toselect = np.random.randint(2)
            if (toselect == 0 and node2crossover1.right != None):

                node2crossover1 = node2crossover1.right
                rangetree1 -=1
            if(toselect==1 and node2crossover1.left != None):

                node2crossover1 = node2crossover1.left
                rangetree1 -= 1
            else:
                rangetree1 -=1
        node2crossover2 = Generations[-1][indOftree2][0].root
        while(rangetree2>0):
            toselect = np.random.randint(2)
            if (toselect == 0 and node2crossover2.right!=None):
                node2crossover2 = node2crossover2.right
                rangetree1 -=1
```

صبا کیانوش