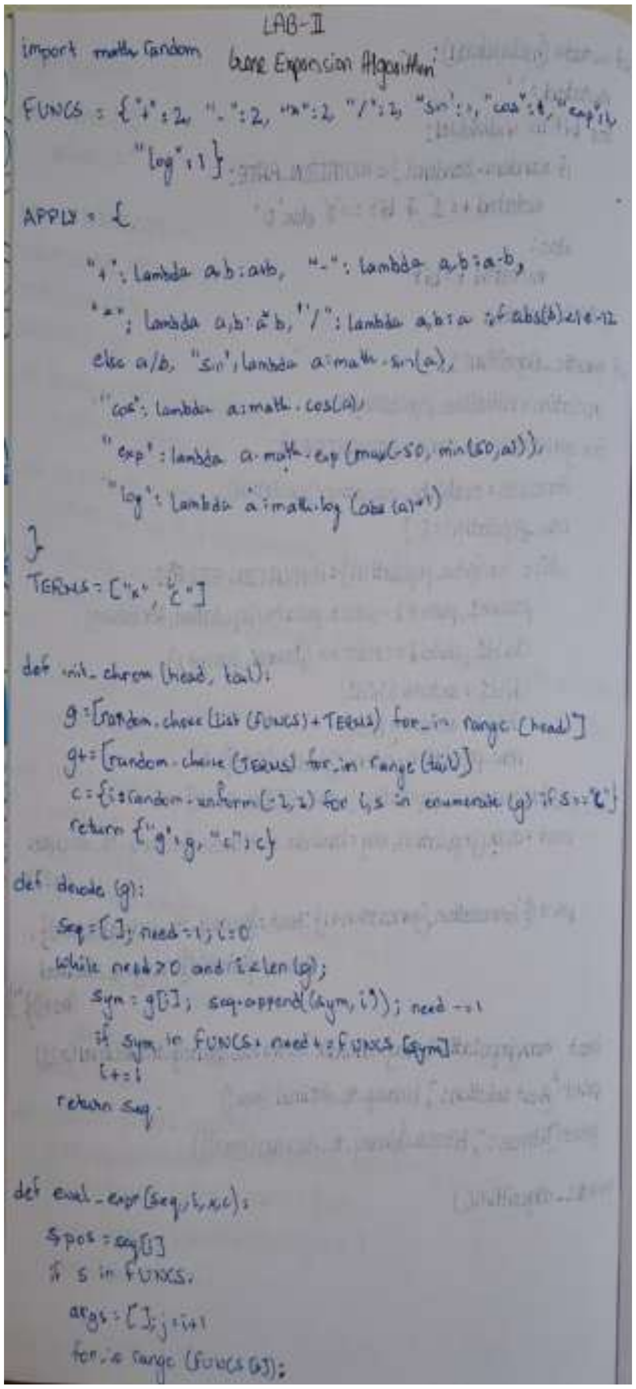
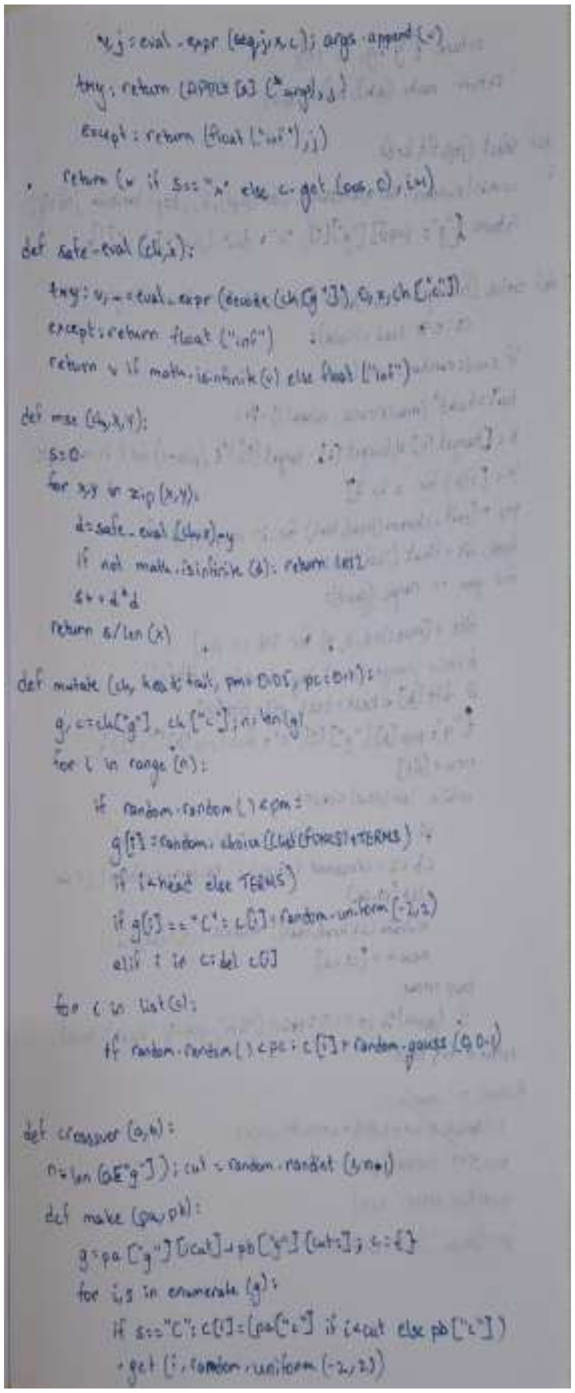
Lab 2

Gene Expression Algorithm



```
LAB-II
import math random        Gene Expension Algorithm

FUNCS = {"+":2, "-":2, "*":2, "/":2, "sin":1, "cos":1, "exp":1,
        "log":1}

APPLY = {
    "+": lambda a,b:a+b,  "-": lambda a,b:a-b,
    "*": lambda a,b:a*b, "/": lambda a,b:a if abs(b)<1e-12
    else a/b, "sin": lambda a:math.sin(a),
    "cos": lambda a:math.cos(a),
    "exp": lambda a:math.exp(max(-50, min(50,a))),
    "log": lambda a:math.log(abs(a)+1)
}
TERMS = ["x", "c"]

def init_chrom(head, tail):
    g=[random.choice(list(FUNCS)+TERMS) for _ in range(head)]
    g+=[random.choice(TERMS) for _ in range(tail)]
    c={i:random.uniform(-2,2) for i,s in enumerate(g) if s=="c"}
    return {"g":g, "c":c}
def decode(g):
    seq=[]; need=1; i=0
    while need>0 and i<len(g):
        sym=g[i]; seq.append((sym,i)); need-=1
        if sym in FUNCS: need+=FUNCS[sym]
        i+=1
    return seq

def eval_expr(seq,i,x,c):
    s,pos=seq[i]
    if s in FUNCS:
        args=[];j=i+1
        for _ in range(FUNCS[s]):
```



```
            v,j=eval_expr(seq,j,x,c); args.append(v)
        try: return (APPLY[s](*args), j)
        except: return (float("inf"),j)
    return (x if s=="x" else c.get(pos,0), i+1)

def safe_eval(ch,x):
    try: v,_=eval_expr(decode(ch["g"]),0,x,ch["c"])
    except: return float("inf")
    return v if math.isfinite(v) else float("inf")

def mse(ch,X,Y):
    s=0
    for x,y in zip(X,Y):
        d=safe_eval(ch,x)-y
        if not math.isfinite(d): return 1e12
        s+=d*d
    return s/len(X)

def mutate(ch, head, tail, pm=0.05, pc=0.1):
    g,c=ch["g"], ch["c"]; n=len(g)
    for i in range(n):
        if random.random()<pm:
            g[i]=random.choice((list(FUNCS)+TERMS)
            if i<head else TERMS)
            if g[i]=="c": c[i]=random.uniform(-2,2)
            elif i in c: del c[i]
    for i in list(c):
        if random.random()<pc: c[i]+=random.gauss(0,0.1)

def crossover(a,b):
    n=len(a["g"]); cut=random.randint(1,n-1)
    def make(pa,pb):
        g=pa["g"][:cut]+pb["g"][cut:]; c={}
        for i,s in enumerate(g):
            if s=="c": c[i]=(pa["c"] if i<cut else pb["c"])
            .get(i, random.uniform(-2,2))
        return {"g":g, "c":c}
    return make (a,b), make (b,a)

def select (pop, fit, k=3):
    i=min(random.sample(range(len(pop)),k), key=lambda j:fit[j])
    return {"g": pop[i]["g"][:], "c": dict (pop[i]["c"])}

def evolve (f, target=(-3,3), pts=64, popn=100, head=10, gens=100,
        cx=0.7, seed=None):
    if seed: random.seed(seed)
    tail=head*(max(FUNCS.values())-1)+1
    X=[target[0]+(target[1]-target[0])*i/pts-1) for i in range(pts)]
    Y=[f(x) for x in X]
    pop=[init_chrom(head,tail) for _ in range(popn)]
    best, fit = float("inf"), None
    for gen in range(gens):
        fits=[mse(ind,X,Y) for ind in pop]
        b=min(range(popn), key=lambda i:fits[i])
        if fits[b] < best: best, fit= fits[b],
        {"g": pop[b]["g"][:], "c": dict(pop[b]["c"])}
        new=[fit]
        while len(new)<popn:
            p1=select(pop,fits); p2=select(pop,fits)
            c1, c2=crossover (p1,p2) if random.random()<cx
            else (p1,p2)
            mutate (c1, head, tail); mutate (c2,head,tail)
            new+=[c1,c2]
        pop=new
        if (gen+1)%20==0: print ("Gen", gen+1, "Best", best)
    return fit, best

if _name_=="_main_":
    f=lambda x:x**3-0.5*x+math.sin(x)
    best,err=evolve(f,gens=100)
    print(Best error:",err)
    print("Preds:",[safe_eval(best,x) for x in [-2,-1,0,1,2]])
```

Code:

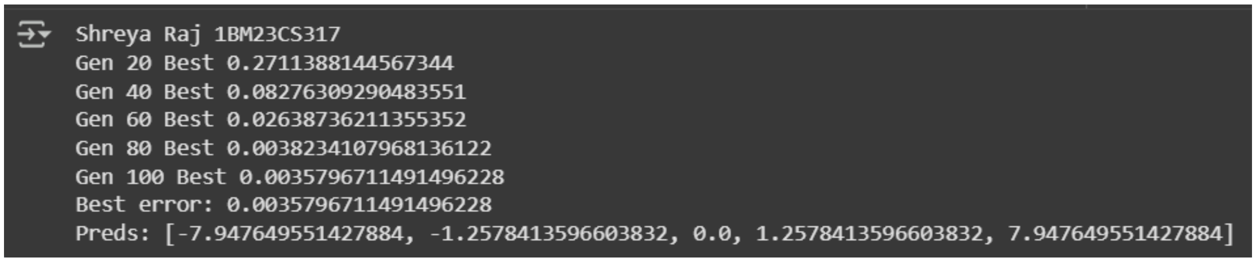
```
print("Shreya Raj 1BM23CS317")
import math, random


# --- Function set ---
FUNCS = {"+":2,"-":2,"*":2,"/":2,"sin":1,"cos":1,"exp":1,"log":1}
APPLY = {
  "+":lambda a,b:a+b, "-":lambda a,b:a-b, "*":lambda a,b:a*b,
  "/":lambda a,b:a if abs(b)<1e-12 else a/b,
  "sin":lambda a:math.sin(a), "cos":lambda a:math.cos(a),
  "exp":lambda a:math.exp(max(-50,min(50,a))),
  "log":lambda a:math.log(abs(a)+1)
}
TERMS=["x","C"]


def init_chrom(head,tail):
  g=[random.choice(list(FUNCS)+TERMS) for _ in range(head)]
  g+=[random.choice(TERMS) for _ in range(tail)]
  c={i:random.uniform(-2,2) for i,s in enumerate(g) if s=="C"}
  return {"g":g,"c":c}


def decode(g):
  seq=[];need=1;i=0
  while need>0 and i<len(g):
    sym=g[i];seq.append((sym,i));need-=1
    if sym in FUNCS: need+=FUNCS[sym]
```

```
        i+=1
    return seq


def eval_expr(seq,i,x,c):
    s,pos=seq[i]
    if s in FUNCS:
        args=[];j=i+1
        for _ in range(FUNCS[s]):
            v,j=eval_expr(seq,j,x,c);args.append(v)
        try:return (APPLY[s](*args),j)
        except: return (float("inf"),j)
    return (x if s=="x" else c.get(pos,0),i+1)


def safe_eval(ch,x):
    try:v,_=eval_expr(decode(ch["g"]),0,x,ch["c"])
    except:return float("inf")
    return v if math.isfinite(v) else float("inf")


def mse(ch,X,Y):
    s=0
    for x,y in zip(X,Y):
        d=safe_eval(ch,x)-y
        if not math.isfinite(d): return 1e12
        s+=d*d
    return s/len(X)
```

```
def mutate(ch,head,tail,pm=0.05,pc=0.1):
  g,c=ch["g"],ch["c"];n=len(g)
  for i in range(n):
    if random.random()<pm:
      g[i]=random.choice((list(FUNCS)+TERMS) if i<head else TERMS)
      if g[i]=="C":c[i]=random.uniform(-2,2)
      elif i in c:del c[i]
  for i in list(c):
    if random.random()<pc:c[i]+=random.gauss(0,0.1)

def crossover(a,b):
  n=len(a["g"]);cut=random.randint(1,n-1)
  def make(pa,pb):
    g=pa["g"][:cut]+pb["g"][cut:];c={}
    for i,s in enumerate(g):
      if s=="C":c[i]=(pa["c"] if i<cut else pb["c"]).get(i,random.uniform(-2,2))
    return {"g":g,"c":c}
  return make(a,b),make(b,a)

def select(pop,fit,k=3):
  i=min(random.sample(range(len(pop)),k),key=lambda j:fit[j])
  return {"g":pop[i]["g"][:],"c":dict(pop[i]["c"])}

def evolve(f,target=(-3,3),pts=64,popn=100,head=10,gens=100,cx=0.7,seed=None):
  if seed:random.seed(seed)
  tail=head*(max(FUNCS.values())-1)+1
```

```
    X=[target[0]+(target[1]-target[0])*i/(pts-1) for i in range(pts)]
    Y=[f(x) for x in X]
    pop=[init_chrom(head,tail) for _ in range(popn)]
    best,fit=float("inf"),None
    for gen in range(gens):
        fits=[mse(ind,X,Y) for ind in pop]
        b=min(range(popn),key=lambda i:fits[i])
        if fits[b]<best:best,fit=fits[b],{"g":pop[b]["g"][:],"c":dict(pop[b]["c"])}
        new=[fit]
        while len(new)<popn:
            p1=select(pop,fits);p2=select(pop,fits)
            c1,c2=crossover(p1,p2) if random.random()<cx else (p1,p2)
            mutate(c1,head,tail);mutate(c2,head,tail)
            new+=[c1,c2]
        pop=new
        if (gen+1)%20==0:print("Gen",gen+1,"Best",best)
    return fit,best

if __name__=="__main__":
    f=lambda x:x**3-0.5*x+math.sin(x)
    best,err=evolve(f,gens=100)
    print("Best error:",err)
    print("Preds:",[safe_eval(best,x) for x in [-2,-1,0,1,2]])
```

Output:



```
Shreya Raj 1BM23CS317
Gen 20 Best 0.2711388144567344
Gen 40 Best 0.08276309290483551
Gen 60 Best 0.02638736211355352
Gen 80 Best 0.0038234107968136122
Gen 100 Best 0.0035796711491496228
Best error: 0.0035796711491496228
Preds: [-7.947649551427884, -1.2578413596603832, 0.0, 1.2578413596603832, 7.947649551427884]
```