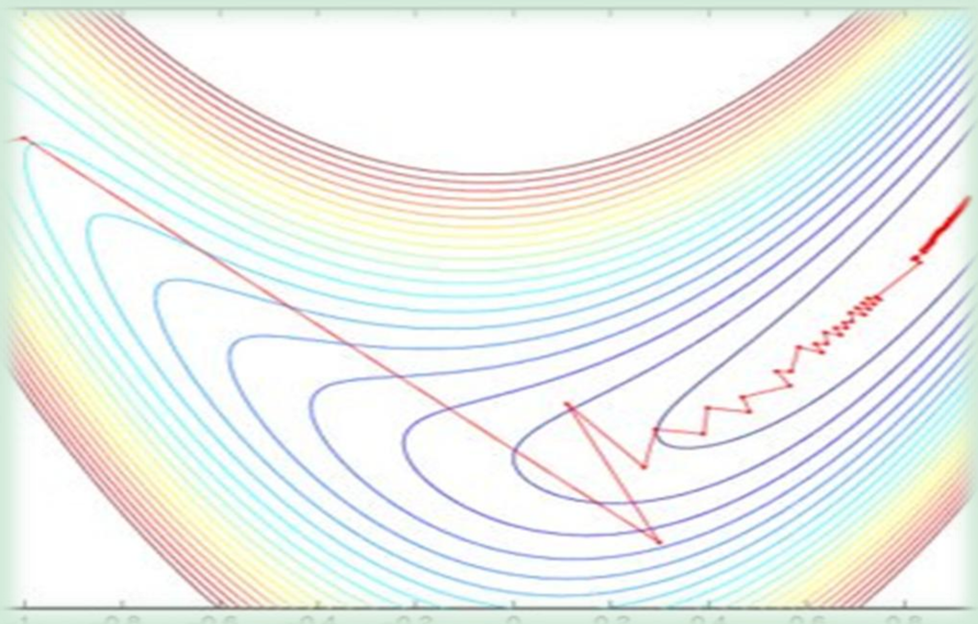
Optimization Techniques

Bonus Task

Report



PRESENTED BY:

Monique Ehab	4928
Hania Mohamed Hani	4697
Hana Hesham Nazmy	4952
Menna Farouk	5088
Ola Ahmed	4607
Noran Amra	4702
Cloudia Kamal	4948

FIRST: PENALTY FUNCTION

A penalty method replaces a constrained optimization problem by a series of unconstrained problems whose solutions ideally converge to the solution of the original constrained problem.

STEPS

1. We convert any constraints into the form (expression) ≤ 0.
2. We start charging a penalty for violating them constraints. Since we're trying to minimize f(x), this means we need to add value when the constraint is violated.
3. We add all the penalty functions on to the original objective function and minimize from there: minimize $T(x) = f(x) + P(x)$
4. We multiply the quadratic loss function by a constant r.
5. We used the gradient descent method and the steepest descent method for the unconstrained minimization.

CODE

```
def move_inequality_constants(ineq):
    l = ineq.lhs
    r = ineq.rhs
    op = ineq.rel_op

    if op.__contains__('<'):
      return l - r
    else:
      return r - l
```

```
while k<int(st):
  h=input()
  h=parse_expr(h.replace('x1','x').replace('x2','y'))
  g.append(move_inequality_constants(h))
  k+=1


start_pt=list(start_pt.replace(',',""))
for p in range(len(start_pt)):
   start_pt[p]=int(start_pt[p])


M=f
for j in g:
  M=M+r*(Max(0,j))**2
```

LECTURE EXAMPLE

An Exterior Point Penalty Function Method

Example: Minimize $f(x_1, x_2) = \frac{1}{3}(x_1+1)^3 + x_2$

subject to

$x_1 \geqslant 1$ and $x_2 \geqslant 0$

- We re-write the constraints as:

  $1 - x_1 \leqslant 0, \quad -x_2 \leqslant 0$

- The function Φ is thus given as:

$$\Phi(X; r_k) = \frac{1}{3}(x_1+1)^3 + x_2 + r_k\left[\max(0, 1-x_1)\right]^2 + r_k\left[\max(0, -x_2)\right]^2$$

Answer:

$r_3 = 10 r_2 = 100$

$$x_1^{(3)} = -1 - 100 + \sqrt{10^4 + 400} = 0.9804$$

$$x_2^{(3)} = \frac{-1}{2(100)} = -0.005$$

PARAMETERS

- Penalty parameter: 1
- Penalty scale: 10
- $f(x)=1/3*(x1+1)^3 + x^2$
- Number of inequality constraints: 2
- Inequality constraints:

x1>=1

x2>=0

SECOND: GRADIENT DESCENT

Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. It is a way to minimize an objective function parameterized by a model's parameters by updating the parameters in the opposite direction of the gradient of the objective function w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum.

STEPS

1. To solve for the gradient, we iterate through our data points and compute the partial derivatives.
2. This new gradient tells us the slope of our cost function at our current position (current parameter values) and the direction we should move to update our parameters.
3. The size of our update is controlled by the learning rate.
4. We subtract because the derivatives point in direction of steepest descent.

CODE

```
delta_k=[sym.diff(M, x),sym.diff(M, y)]
print("dΦ{}=")
print(delta_k)
i=0
while i<iterations:
  delta_1=[delta_k[0].subs(x, start_pt[0]).subs(y, start_pt[1]).subs(r,r_k),delta_k[1].subs(x, start_pt[0]).subs(y, start_pt[1]).subs(r,r_k) ]
  X=[m - n for m,n in zip(start_pt,[j * e for j in delta_1])]
  for m in range(gd_iterations-1):
    delta_1=[delta_k[0].subs(x, X[0]).subs(y, X[1]).subs(r,r_k),delta_k[1].subs(x, X[0]).subs(y, X[1]).subs(r,r_k) ]
    X=[m - n for m,n in zip(X,[j * e for j in delta_1])]
  t=0
  for k in range(int(st)):
     if g[k].subs(x,X[0]).subs(y,X[1])<=0:
        t+=1
     else:
      break
  r_k*=c
  if t==int(int(st)):
    break
  i+=1
print("fmin= {} at point {} ".format(f.subs(x, X[0]).subs(y,X[1]), X))
```

PARAMETERS

- Learning rate: 0.1
- Number of iterations for gradient descent method: 10
- Starting points: 0,0

SAMPLE RUN

```
Enter penalty parameter: 1
Enter penalty scale: 10
Enter the learning rate: 0.1
Enter number of iterations for gradient descent method: 10
Enter starting points: 0,0
f(x)=1/3*(x1+1)**3+x2
Enter number of inequality constraints: 2
x1>=1
x2>=0
Φ{}=
r*Max(0, -y)**2 + r*Max(0, -x + 1)**2 + y + (x + 1)**3/3
dΦ{}=
[-2*r*Heaviside(-x + 1)*Max(0, -x + 1) + (x + 1)**2, -2*r*Heaviside(-y)*Max(0, -y) + 1]
fmin= 2.96867560865580 at point [1.07281877151427, 0]
```

THIRD: STEEPEST DESCENT

The steepest descent method is the simplest of the gradient methods for optimization in n variables. If we want to minimize a function F(x) and if our current trial point is x then we can expect to find better points by moving away from x along the direction which causes F to decrease most rapidly. This direction of steepest descent is given by the negative gradient.

STEPS

1. We set $pk = -\nabla F(xk)$
2. Set $\alpha k = \text{argmin } \phi(\alpha) = f(xk) - \alpha gk$
3. $xk+1 = xk - \alpha k*gk$
4. Compute $gk+1 = \nabla f(xk+1)$

CODE

```
delta_k=[sym.diff(f, x),sym.diff(f, y)]
delta_1=[delta_k[0].subs(x, start_pt[0]).subs(y, start_pt[1]),delta_k[1].subs(x, start_pt[0]).subs(y, start_pt[1]) ]

lambdaval1= start_pt[0] - delta_1[0]*l
lambdaval2=start_pt[1] - delta_1[1]*l


find=sym.diff(f.subs(x, lambdaval1).subs(y, lambdaval2),l)

sol=solve(find)
print("Lambda value:")
print(sol)

X=[m - n for m,n in zip(start_pt,[j * sol[0] for j in delta_1])]

for m in range(gd_iterations-1):
  delta_1=[delta_k[0].subs(x, X[0]).subs(y, X[1]),delta_k[1].subs(x, X[0]).subs(y, X[1]) ]
  lambdaval1= X[0] - delta_1[0]*l
  lambdaval2=X[1] - delta_1[1]*l
  find=sym.diff(f.subs(x, lambdaval1).subs(y, lambdaval2),l)
  sol=solve(find)

  X=[m - n for m,n in zip(X,[j * sol[0] for j in delta_1])]
```

LECTURE EXAMPLE

Steepest Descent (Cauchy's) Method

Example: Minimize $f(x,y) = x - y + 2x^2 + 2xy + y^2$ starting from the point $X_1 = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}$ and using three iterations of the steepest descent method.

Answer:

$$X_4 = X_3 - \lambda^* \nabla f_3 = \begin{Bmatrix} -0.8 \\ 1.2 \end{Bmatrix} + 1 \begin{Bmatrix} -0.2 \\ 0.2 \end{Bmatrix} = \begin{Bmatrix} -1.0 \\ 1.4 \end{Bmatrix}$$

PARAMETERS

- Number of iterations for steepest descent method: 3
- Starting points: 0,0
- f(x)=x1-x2+2*x1**2+2*x1*x2+x2**2
- Lambda value: 1

```
Enter number of iterations for steepest descent method: 3
Enter starting points: 0,0
f(x)=x1-x2+2*x1**2+2*x1*x2+x2**2
Lambda value:
[1]
fmin= -31/25 at point [-1, 7/5]
```

PENALTY FUNCION + STEEPEST DESCENT CODE

```
delta_k=[sym.diff(M, x),sym.diff(M, y)]

i=0
while i<iterations:
  delta_1=[delta_k[0].subs(x, start_pt[0]).subs(y, start_pt[1]).subs(r,r_k),delta_k[1].subs(x, start_pt[0]).subs(y, start_pt[1]).subs(r,r_k) ]
  lambdaval1= start_pt[0] - delta_1[0]*l
  lambdaval2=start_pt[1] - delta_1[1]*l
  find=sym.diff(f.subs(x, lambdaval1).subs(y, lambdaval2).subs(r,r_k),l)
  sol=solve(find)
  f1=f.subs(x, start_pt[0] - delta_1[0]*sol[0]).subs(y, start_pt[1] - delta_1[1]*sol[0]).subs(r,r_k)
  f2=f.subs(x, start_pt[0] - delta_1[0]*sol[1]).subs(y, start_pt[1] - delta_1[1]*sol[1]).subs(r,r_k)
  if (f1<f2):
    lam=sol[0]
  else:
    lam=sol[1]
  X=[m - n for m,n in zip(start_pt,[j * lam for j in delta_1])]

  for m in range(gd_iterations-1):
    delta_1=[delta_k[0].subs(x, X[0]).subs(y, X[1]).subs(r,r_k),delta_k[1].subs(x, X[0]).subs(y, X[1]).subs(r,r_k) ]
    lambdaval1= X[0] - delta_1[0]*l
    lambdaval2=X[1] - delta_1[1]*l
    find=sym.diff(f.subs(x, lambdaval1).subs(y, lambdaval2).subs(r,r_k),l)
    sol=solve(find)
    f1=f.subs(x, X[0] - delta_1[0]*sol[0]).subs(y, X[1] - delta_1[1]*sol[0]).subs(r,r_k)
    f2=f.subs(x, X[0] - delta_1[0]*sol[1]).subs(y, X[1] - delta_1[1]*sol[1]).subs(r,r_k)
    if (f1<f2):
      lam=sol[0]
    else:
      lam=sol[1]
    X=[m - n for m,n in zip(X,[j * lam for j in delta_1])]
  t=0
  for k in range(int(st)):
     if g[k].subs(x,X[0]).subs(y,X[1])<=0:
        t+=1
     else:
      break
  r_k*=c
  if t==int(int(st)):
    break
  i+=1
print("fmin= {} at point {} ".format(f.subs(x, X[0]).subs(y,X[1]), X))
```

SAMPLE RUN

```
Enter penalty parameter: 1
Enter penalty scale: 10
Enter number of iterations for steepest descent method: 15
Enter starting points: 0,0
f(x)=1/3*(x1+1)**3+x2
Enter number of inequality constraints: 2
x1>=1
x2>=0
Φ{}=
r*Max(0, -y)**2 + r*Max(0, -x + 1)**2 + y + (x + 1)**3/3
dΦ{}=
[-2*r*Heaviside(-x + 1)*Max(0, -x + 1) + (x + 1)**2, -2*r*Heaviside(-y)*Max(0, -y) + 1]
fmin= 2.98376591497785 at point [1.01663604932899, 0.249999999999999]
```