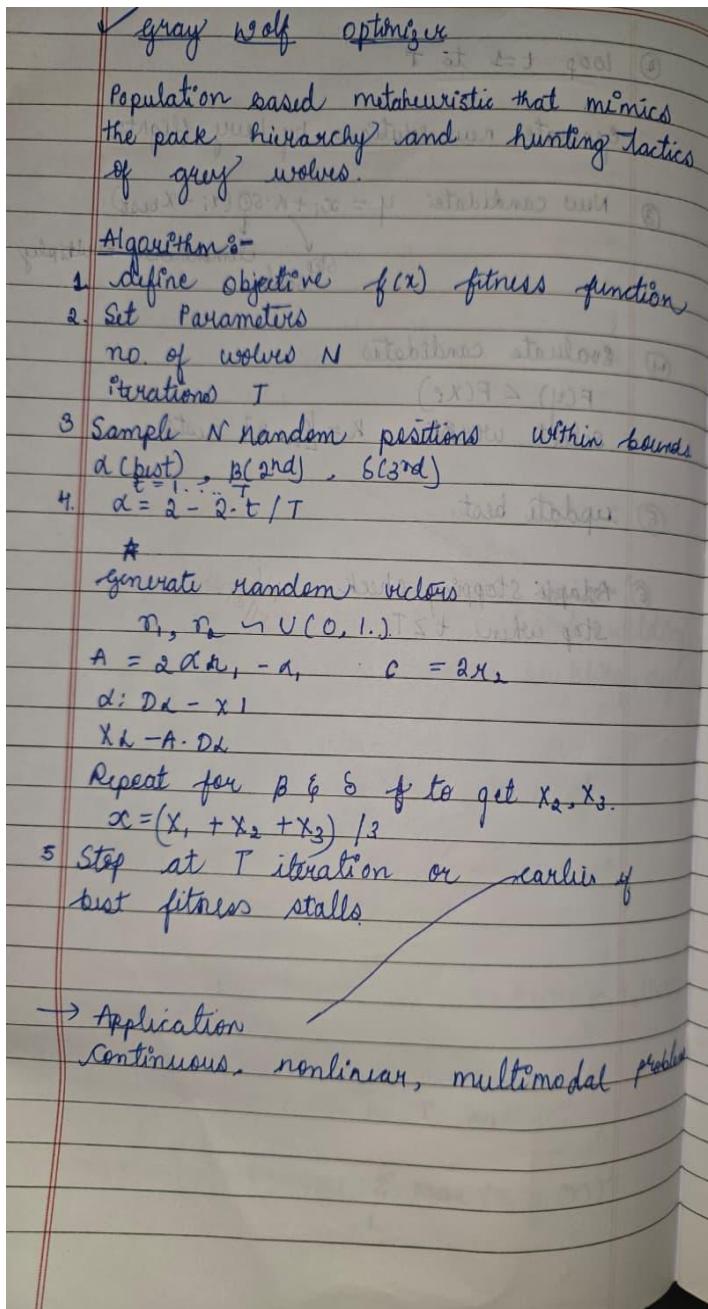
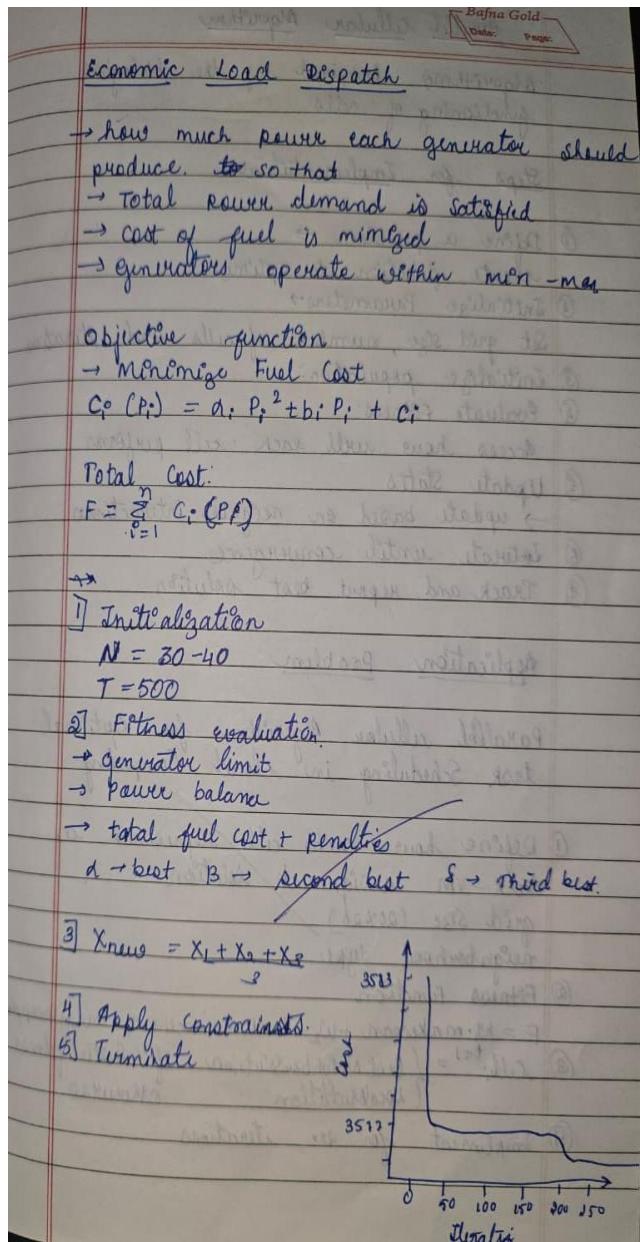


## Program 5

Grey Wolf Optimizer (GWO) Using the Grey Wolf Optimizer (GWO), we aim to find the shortest, obstacle-free path by modeling the search agents (wolves) to iteratively converge toward the best position (path node) in the environment. The algorithm simulates the grey wolves' hunting hierarchy and encircling behavior to efficiently navigate the space from the start point.

### Algorithm:





Code:

```
import numpy as np

a = np.array([0.003, 0.005, 0.001])
b = np.array([7, 8, 6])
c = np.array([100, 120, 150])

Pmin = np.array([50, 50, 50])
Pmax = np.array([200, 150, 180])

Pd = 350
```

```

def cost(P):
    return np.sum(a * P**2 + b * P + c)

def penalty(P):
    total = np.sum(P)
    return 1000 * (abs(total - Pd))

def fitness(P):
    return cost(P) + penalty(P)

def GWO(num_wolves=30, max_iter=200):
    dim = 3
    lb, ub = Pmin, Pmax

    wolves = np.random.uniform(lb, ub, (num_wolves, dim))

    alpha, beta, delta = None, None, None

    for t in range(max_iter):
        for i in range(num_wolves):
            f = fitness(wolves[i])

            if alpha is None or f < fitness(alpha):
                delta = beta
                beta = alpha
                alpha = wolves[i].copy()
            elif beta is None or f < fitness(beta):
                delta = beta
                beta = wolves[i].copy()
            elif delta is None or f < fitness(delta):
                delta = wolves[i].copy()

            a_t = 2 - 2 * (t / max_iter)

            for i in range(num_wolves):
                for j in range(dim):
                    r1, r2 = np.random.rand(), np.random.rand()

```

```

A1 = 2 * a_t * r1 - a_t
C1 = 2 * r2

D_alpha = abs(C1 * alpha[j] - wolves[i][j])
X1 = alpha[j] - A1 * D_alpha

r1, r2 = np.random.rand(), np.random.rand()
A2 = 2 * a_t * r1 - a_t
C2 = 2 * r2
D_beta = abs(C2 * beta[j] - wolves[i][j])
X2 = beta[j] - A2 * D_beta

r1, r2 = np.random.rand(), np.random.rand()
A3 = 2 * a_t * r1 - a_t
C3 = 2 * r2
D_delta = abs(C3 * delta[j] - wolves[i][j])
X3 = delta[j] - A3 * D_delta

wolves[i][j] = (X1 + X2 + X3) / 3

wolves[i] = np.clip(wolves[i], lb, ub)

if t % 50 == 0:
    print(f"Iteration {t} | Best Cost: {cost(alpha)})")

return alpha, cost(alpha)

```

```

best_P, best_cost = GWO()
print("\nBest Power Output:", best_P)
print("Minimum Cost:", best_cost)

```

### Output:

```

...
... Iteration 0 | Best Cost: 2975.9786281503466
... Iteration 50 | Best Cost: 2864.070964959683
... Iteration 100 | Best Cost: 2864.070964959683
... Iteration 150 | Best Cost: 2864.070964959683

Best Power Output: [ 53.18357901 116.81895164 180.
Minimum Cost: 2865.955482677795

```