Here is a detailed line-by-line explanation of the MATLAB code for **ANN-based Droop Control** and **GWO for MPPT**:

ANN-based Droop Control

Step 1: Define Training Data

```
X_train = [0.1, 0.2; 0.3, 0.4; 0.5, 0.6; 0.7, 0.8; 0.9, 1.0]; % Inputs: P and Q
y_train = [50, 220; 49.5, 219; 49, 218; 48.5, 217; 48, 216]; % Outputs: f and V
```

- **`X_train`**: Input matrix where each row represents active power (P) and reactive power (Q).
- **`y_train`**: Output matrix where each row represents frequency (f) and voltage (V).

Step 2: Create and Configure the Neural Network

```
net = feedforwardnet(5, 'trainlm'); % Initialize a feedforward neural network with 5 hidden neurons
```

- **`feedforwardnet(5, 'trainlm')`**: Creates a neural network with:
  - One hidden layer containing 5 neurons.
  - The Levenberg-Marquardt algorithm (`'trainlm'`) for training.

```
net.layers{1}.transferFcn = 'tansig'; % Hidden layer uses tan-sigmoid transfer function
net.layers{2}.transferFcn = 'purelin'; % Output layer uses a linear transfer function
```

- **`transferFcn`**: Specifies the activation function for each layer.
  - `'tansig'`: A nonlinear tan-sigmoid function for the hidden layer.
  - `'purelin'`: A linear function for the output layer.

```
net.trainParam.epochs = 1000; % Set the maximum number of training iterations
net.trainParam.lr = 0.01; % Set the learning rate
net.trainParam.goal = 1e-6; % Set the performance goal
```

- **`trainParam`**: Sets the training parameters:
  - **`epochs`**: Maximum iterations.
  - **`lr`**: Learning rate.
  - **`goal`**: Minimum mean squared error (MSE) the network aims to achieve.

Step 3: Train the Neural Network

```
[net, training_info] = train(net, X_train', y_train'); % Train the network
```

- **`train()`**: Trains the network using the input (`X_train'`) and output (`y_train'`) data (transposed because MATLAB expects column vectors).
- **`training_info`**: Stores training performance metrics.

Step 4: Visualize Training Performance

```
figure;
plotperform(training_info); % Plot performance over epochs
```

- **`plotperform()`**: Displays how the performance (error) decreases over epochs during training.

Step 5: Test the Neural Network

```
X_test = [0.25, 0.35]'; % Define test input
```

```
prediction = net(X_test); % Use the trained network to predict output
fprintf('Predicted Frequency: %.2f Hz, Predicted Voltage: %.2f V\n', prediction(1), prediction(2));
```

- **`net(X_test)`**: Uses the trained network to predict frequency and voltage for the given power input.
- **`fprintf()`**: Prints the predicted frequency and voltage.

Step 6: Save the Trained Network

```
save('trained_ann_droop.mat', 'net'); % Save the trained network to a file
```

- Saves the trained network model for future use.

GWO for MPPT

Step 1: Define the Objective Function

```
objFunc = @(x) -((x(1) - 5)^2 + (x(2) - 3)^2); % Quadratic objective function
```

- **`objFunc`**: A power curve where the goal is to find the maximum (negative sign used to convert it to a minimization problem).

Step 2: Define GWO Parameters

```
dim = 2; % Number of variables
lb = 0; % Lower bound
ub = 10; % Upper bound
population_size = 10; % Number of wolves
max_iter = 200; % Maximum iterations
```

- **`dim`**: Dimension of the search space (active and reactive power here).
- **`lb`, `ub`**: Define the bounds for the search space.
- **`population_size`**: Number of wolves in the population.
- **`max_iter`**: Number of iterations for the optimization process.

Step 3: Initialize Grey Wolves

```
wolves = lb + (ub - lb) .* rand(population_size, dim); % Initialize positions randomly
alpha = zeros(1, dim);
beta = zeros(1, dim);
delta = zeros(1, dim);
alpha_score = inf; beta_score = inf; delta_score = inf;
```

- **`wolves`**: Matrix where each row represents the position of a wolf in the search space.
- **`alpha`, `beta`, `delta`**: The top 3 wolves based on fitness.
- **`alpha_score`, `beta_score`, `delta_score`**: Fitness scores of the top 3 wolves.

Step 4: Optimization Loop

```
for t = 1:max_iter
```

- Main optimization loop that runs for `max_iter` iterations.

```
for i = 1:population_size
    score = objFunc(wolves(i, :)); % Evaluate the fitness of each wolf
    if score < alpha_score
        delta_score = beta_score; delta = beta;
        beta_score = alpha_score; beta = alpha;
        alpha_score = score; alpha = wolves(i, :);
    elseif score < beta_score
```

```
        delta_score = beta_score; delta = beta;
        beta_score = score; beta = wolves(i, :);
    elseif score < delta_score
        delta_score = score; delta = wolves(i, :);
    end
end
```

- Evaluates fitness for each wolf and updates `alpha`, `beta`, and `delta`.

```
a = 2 - t * (2 / max_iter); % Linearly decreasing parameter
for i = 1:population_size
    for j = 1:dim
        % Calculate influence of alpha, beta, delta wolves
        r1 = rand(); r2 = rand();
        A1 = 2 * a * r1 - a;
        C1 = 2 * r2;
        D_alpha = abs(C1 * alpha(j) - wolves(i, j));
        X1 = alpha(j) - A1 * D_alpha;

        r1 = rand(); r2 = rand();
        A2 = 2 * a * r1 - a;
        C2 = 2 * r2;
        D_beta = abs(C2 * beta(j) - wolves(i, j));
        X2 = beta(j) - A2 * D_beta;

        r1 = rand(); r2 = rand();
        A3 = 2 * a * r1 - a;
        C3 = 2 * r2;
        D_delta = abs(C3 * delta(j) - wolves(i, j));
        X3 = delta(j) - A3 * D_delta;

        % Update position of the wolf
        wolves(i, j) = (X1 + X2 + X3) / 3;
        wolves(i, j) = max(min(wolves(i, j), ub), lb); % Bound the position
    end
end
```

- Updates the position of each wolf based on influences from the top 3 wolves (`alpha`, `beta`, `delta`).

```
if mod(t, 10) == 0
    fprintf('Iteration %d: Best Power = %.2f at [%f, %f]\n', t, -alpha_score, alpha(1), alpha(2));
end
```

- Prints progress every 10 iterations.

Step 5: Results

```
fprintf('Optimal Point: %.2f, %.2f\n', alpha(1), alpha(2));
fprintf('Optimal Power: %.2f\n', -alpha_score);
```

- Displays the optimal point and power.

Step 6: Visualization

```
figure;
plot(1:max_iter, -alpha_score * ones(max_iter, 1), '-o');
xlabel('Iteration'); ylabel('Best Power');
title('GWO Optimization Progress'); grid on;
```

- Plots the optimization progress over iterations.

Let me know if you need further clarification!