```
import networkx
import matplotlib.pyplot as plt
import numpy as np
import powerlaw


class NonlinearNetwork:
    def __init__(self, N, m, alpha, beta):
        self.N = N
        self.m = m
        self.alpha = alpha
        self.beta = beta
        self.G = networkx.barabasi_albert_graph(m, 1)
        self.degree_hist = networkx.degree_histogram(self.G)
        self.new_node_index = m

    def nonlinear_preferential_attachment_network(self):
        while len(self.G.nodes()) < self.N:
            weights = [(degree ** self.alpha) * (1 + self.beta * np.random.randn()) for
degree in
                        dict(self.G.degree()).values()]
            targets = np.random.choice(self.G.nodes(), size=self.m, replace=False,
p=weights / np.sum(weights))
            self.G.add_node(self.new_node_index)
            self.G.add_edges_from([(self.new_node_index, t) for t in targets])
            self.degree_hist = networkx.degree_histogram(self.G)
            self.new_node_index += 1
        return self.G

    def degree_calculator(self):
        degrees = [val for (node, val) in self.G.degree()]
        fit = powerlaw.Fit(degrees)
        print(
            f"alpha={fit.alpha}, xmin={fit.xmin}, sigma={fit.sigma}, degree = {2 *
len(self.G.degree()) / len(self.G.nodes())}")
        return fit.alpha, fit.xmin, fit.sigma

    def degree_distribution(self):
        plt.figure(figsize=(8, 8))
        plt.plot(self.degree_hist)
        plt.show()

    def show(self):
        plt.figure(figsize=(8, 8))
```

```
networkx.draw(self.G, with_labels=True)
plt.show()
```