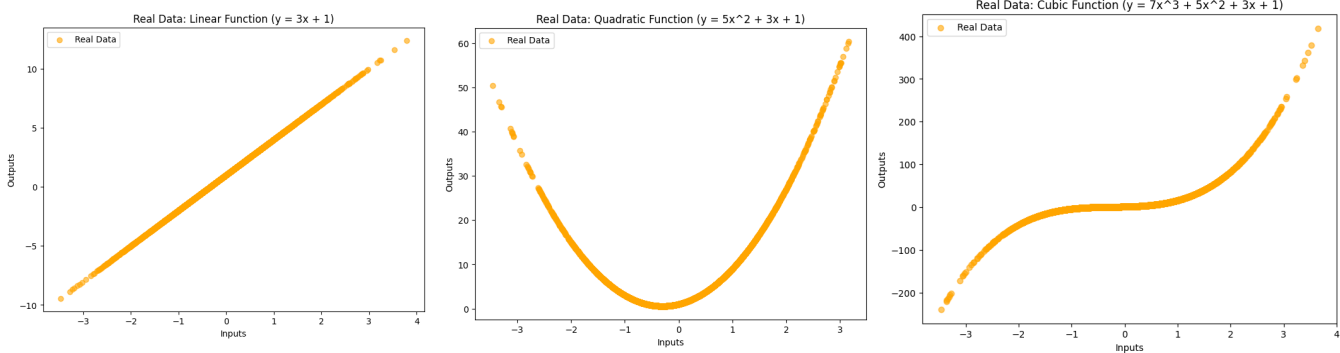


# Assignment-5 Report

Teoman Kaman

1. Generate the real data for each equation and plot it. For this question my real data plots are:



2. Create a GAN model with a Generator and Discriminator. For the classes gan is like this:

```
class Generator(nn.Module):
    def __init__(self, latent_dim, input_dim, output_dim):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(latent_dim + input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 64),
            nn.ReLU(),
            nn.Linear(64, output_dim)
        )

    def forward(self, z, x):
        combined = torch.cat((z, x), dim=1)
        return self.model(combined)

# Define the discriminator
class Discriminator(nn.Module):
    def __init__(self, input_dim):
        super(Discriminator, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.LeakyReLU(0.2),
            nn.Linear(64, 64),
            nn.LeakyReLU(0.2),
            nn.Linear(64, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.model(x)
```

3. Train your generator and discriminator to produce their expected behavior. For this question my logs looks like this:

**Linear:**

Epoch 0/10000, D Loss: 1.4147, G Loss: 0.7268  
Epoch 1000/10000, D Loss: 1.3713, G Loss: 0.6964  
Epoch 2000/10000, D Loss: 1.3761, G Loss: 0.6987  
Epoch 3000/10000, D Loss: 1.3808, G Loss: 0.6969  
Epoch 4000/10000, D Loss: 1.3835, G Loss: 0.6921  
Epoch 5000/10000, D Loss: 1.3843, G Loss: 0.6994  
Epoch 6000/10000, D Loss: 1.3828, G Loss: 0.6952  
Epoch 7000/10000, D Loss: 1.3832, G Loss: 0.6925  
Epoch 8000/10000, D Loss: 1.3821, G Loss: 0.6952  
Epoch 9000/10000, D Loss: 1.3825, G Loss: 0.6901

---

### Quadratic:

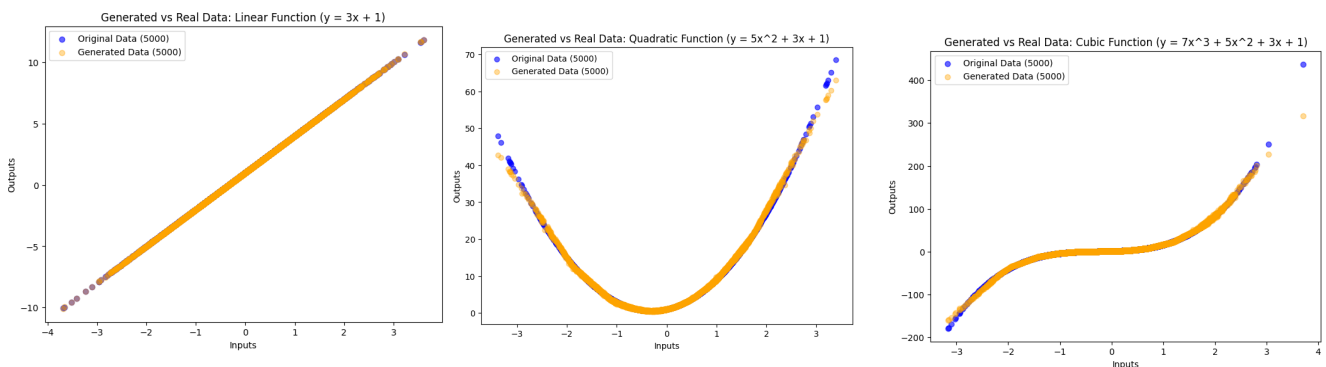
Epoch 0/10000, D Loss: 1.3816, G Loss: 0.7617  
Epoch 1000/10000, D Loss: 1.0653, G Loss: 0.9943  
Epoch 2000/10000, D Loss: 1.2859, G Loss: 0.7429  
Epoch 3000/10000, D Loss: 1.3206, G Loss: 0.7688  
Epoch 4000/10000, D Loss: 1.3385, G Loss: 0.7271  
Epoch 5000/10000, D Loss: 1.3428, G Loss: 0.7442  
Epoch 6000/10000, D Loss: 1.3573, G Loss: 0.7145  
Epoch 7000/10000, D Loss: 1.3653, G Loss: 0.6997  
Epoch 8000/10000, D Loss: 1.3703, G Loss: 0.7045  
Epoch 9000/10000, D Loss: 1.3739, G Loss: 0.6902

---

### Cubic:

Epoch 0/25000, D Loss: 1.2246, G Loss: 0.6973  
Epoch 1000/25000, D Loss: 1.1501, G Loss: 0.9648  
Epoch 2000/25000, D Loss: 1.0695, G Loss: 1.0313  
Epoch 3000/25000, D Loss: 1.1503, G Loss: 0.9626  
Epoch 4000/25000, D Loss: 1.2478, G Loss: 0.8411  
Epoch 5000/25000, D Loss: 1.2948, G Loss: 0.7552  
Epoch 6000/25000, D Loss: 1.3008, G Loss: 0.7544  
Epoch 7000/25000, D Loss: 1.3224, G Loss: 0.7581  
Epoch 8000/25000, D Loss: 1.3461, G Loss: 0.7153  
Epoch 9000/25000, D Loss: 1.3376, G Loss: 0.7587  
Epoch 10000/25000, D Loss: 1.3555, G Loss: 0.7155  
Epoch 11000/25000, D Loss: 1.3473, G Loss: 0.7435  
Epoch 12000/25000, D Loss: 1.3547, G Loss: 0.7187  
Epoch 13000/25000, D Loss: 1.3588, G Loss: 0.7010  
Epoch 14000/25000, D Loss: 1.3643, G Loss: 0.7130  
Epoch 15000/25000, D Loss: 1.3605, G Loss: 0.7064  
Epoch 16000/25000, D Loss: 1.3744, G Loss: 0.7102  
Epoch 17000/25000, D Loss: 1.3759, G Loss: 0.6908  
Epoch 18000/25000, D Loss: 1.3726, G Loss: 0.6930  
Epoch 19000/25000, D Loss: 1.3763, G Loss: 0.7073  
Epoch 20000/25000, D Loss: 1.3700, G Loss: 0.7029  
Epoch 21000/25000, D Loss: 1.3698, G Loss: 0.6838  
Epoch 22000/25000, D Loss: 1.3838, G Loss: 0.6620  
Epoch 23000/25000, D Loss: 1.3714, G Loss: 0.7012  
Epoch 24000/25000, D Loss: 1.3769, G Loss: 0.6924

4. After training the GAN use the trained generator to generate fake samples. Once you have fake samples, plot real (total 5000) and fake samples (total 5000). For this question my plots for all the functions are:



### 5. Summary:

The linear function performed the best because its simplicity made it easy for the model to learn. The quadratic function also fit well, but there were small differences at the edges due to its increased complexity. The cubic function had the most difficulty, with larger gaps between the real

and generated data, especially for extreme inputs so I tried it with more epochs and it made it better compared to other models in terms of epochs.

These results show that simpler models like the linear function are easier to train, while quadratic and cubic functions require more effort to capture their complexity. Overall, the performance matches the expected behavior of each function.