UNIVERSIDADE FEDERAL
DE MINAS GERAIS

ATIVIDADE AVALIATIVA 1
APROXIMAÇÃO POLINOMIAL

ALUNO: FELIPPE VELOSO MARINHO
MATRÍCULA: 2021072260
DISCIPLINA: REDES NEURAIS ARTIFICIAIS

1 -

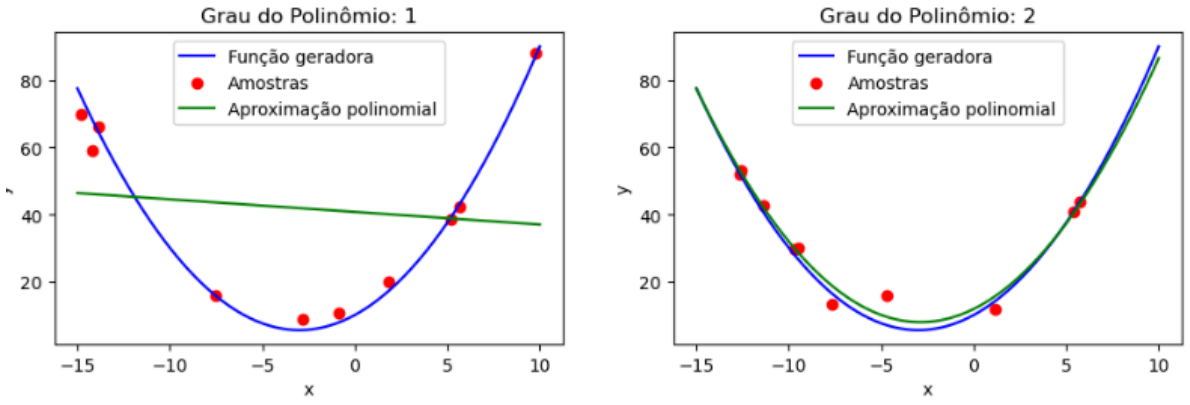
O objetivo da atividade era obter aproximações polinomiais da função geradora $fg(x)$ = 1 2 x 2 + 3x + 10 somadas com um ruído gaussiano $N(mean = 0, sd = 4)$ amostradas entre $x = -15$ e $x = 10$, com um número de amostras $N = 10$ e grau do polinômio variando entre $p = 1$ a $p = 8$.
Para isso, após importar as funções utilizadas no código, foram definidas a função geradora e os parâmetros dados no enunciado.

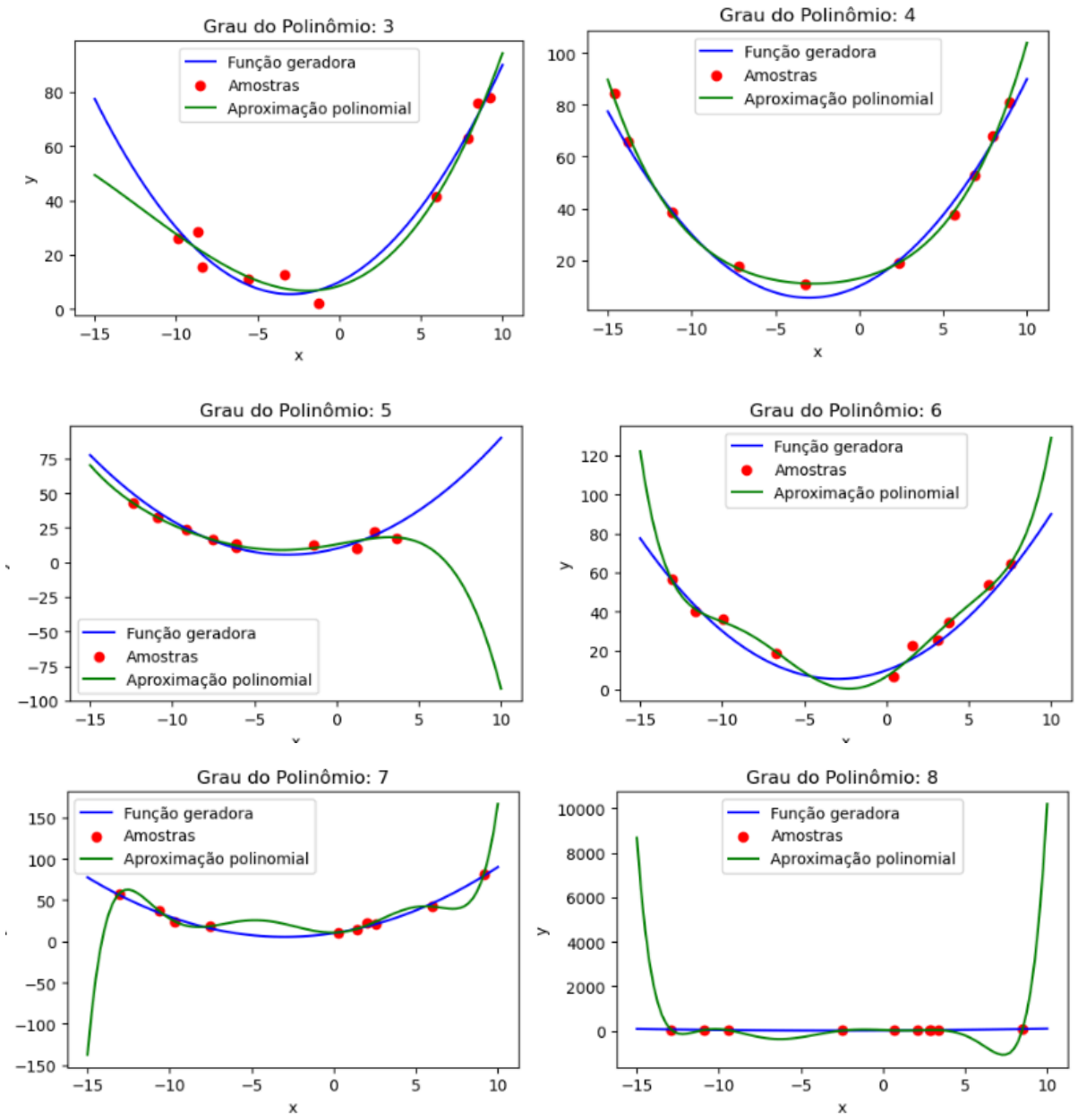
```
# Definindo função geradora
def f(x):
    return 0.5 * x**2 + 3 * x + 10

# parâmetros do ruído gaussiano
mean_noise, sd_noise = 0, 4

N = 100 # número de amostras
p_values = range(1, 9) # Definir graus dos polinômios a serem ajustados (de 1 a 8)
x_values = np.linspace(-15, 10, 100) # amostradas entre x = -15 e x = 10

# Parâmetros para a geração dos dados
np.random.seed(99)

# Gerando dados e plotando para cada grau de polinômio
plt.figure(figsize=(15, 10))
```

Para utilizar diferentes graus de polinômios, foi utilizado o loop for abaixo, onde cada iteração representa um grau específico. Para cada grau de polinômio, amostras aleatórias são geradas dentro de um intervalo definido, seguidas pela adição de um ruído gaussiano. Depois, como comentado no código, ajustamos aos dados da amostra utilizando as classes importadas da biblioteca ‘scikit-learn’ e plotamos os gráficos.

```python
for i, p in enumerate(p_values, 1):
    # Gerando amostras
    x_samples = np.random.uniform(-15, 10, N)
    y_samples = f(x_samples) + np.random.normal(mean_noise, sd_noise, N)

    # Ajustando o polinômio aos dados
    poly_features = PolynomialFeatures(degree=p) # função importada da biblioteca
    X_poly = poly_features.fit_transform(x_samples.reshape(-1, 1))
    lin_reg = LinearRegression()
    lin_reg.fit(X_poly, y_samples)

    # Preparando dados para plotagem
    x_plot = np.linspace(-15, 10, 100)
    X_plot_poly = poly_features.transform(x_plot.reshape(-1, 1))
    y_plot = lin_reg.predict(X_plot_poly)

    # Plotando
    plt.subplot(3, 3, i)
    plt.plot(x_values, f(x_values), label='Função geradora', color='blue')
    plt.scatter(x_samples, y_samples, label='Amostras', color='red')
    plt.plot(x_plot, y_plot, label='Aproximação polinomial', color='green')
    plt.title(f'Grau do Polinômio: {p}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()

plt.tight_layout()
plt.show()
```

Para cada uma das 8 aproximações, o gráfico com a função geradora, as amostras e a curva do polinômio obtido está exibido abaixo.





2 -

"Overfitting: Ocorre quando o modelo se ajusta muito bem aos dados de treinamento, mas não consegue generalizar bem para novos dados. Isso geralmente ocorre quando o modelo é muito complexo em relação à quantidade de dados disponíveis.

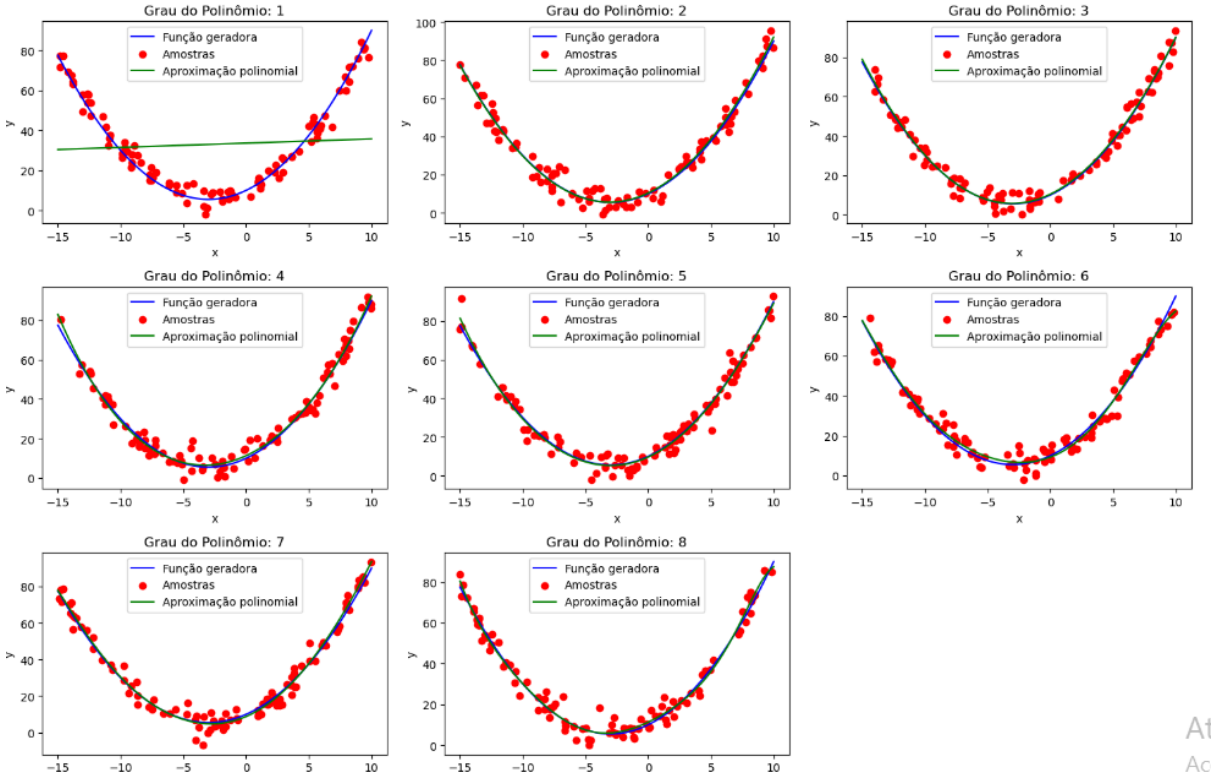
Underfitting: Ocorre quando o modelo é muito simples para capturar a complexidade dos dados, resultando em um desempenho ruim tanto nos dados de treinamento quanto nos novos dados."

- O modelo com grau de polinômio 1 é um exemplo de underfitting, pois é visível que o modelo de aproximação não conseguiu se adaptar aos dados do treinamento.

- Os modelos de grau de polinômio 2 até 4 obtiveram melhores comportamentos e obtiveram uma aproximação consistente da função geradora. Sendo assim, não ocorreu over nem underfitting para estes casos.

- Os últimos 4 casos obtiveram resultados estranhos e é possível dizer que o modelo se ajustou bem em relação às amostras de entrada, porém obteve severa dificuldade na previsão de novos resultados. Sendo assim, se enquadra m como Overfitting.

3 -



- Como esperado, com o maior número de amostras o desempenho da aproximação polinomial foi melhor. Com mais dados, o modelo pode capturar melhor a relação entre as variáveis. Tirando o polinômio de grau 1 (underfitting), todos os resultados obtiveram aproximações bem satisfatórias, sendo bem próximas a função geradora.