0.0.1 Imports

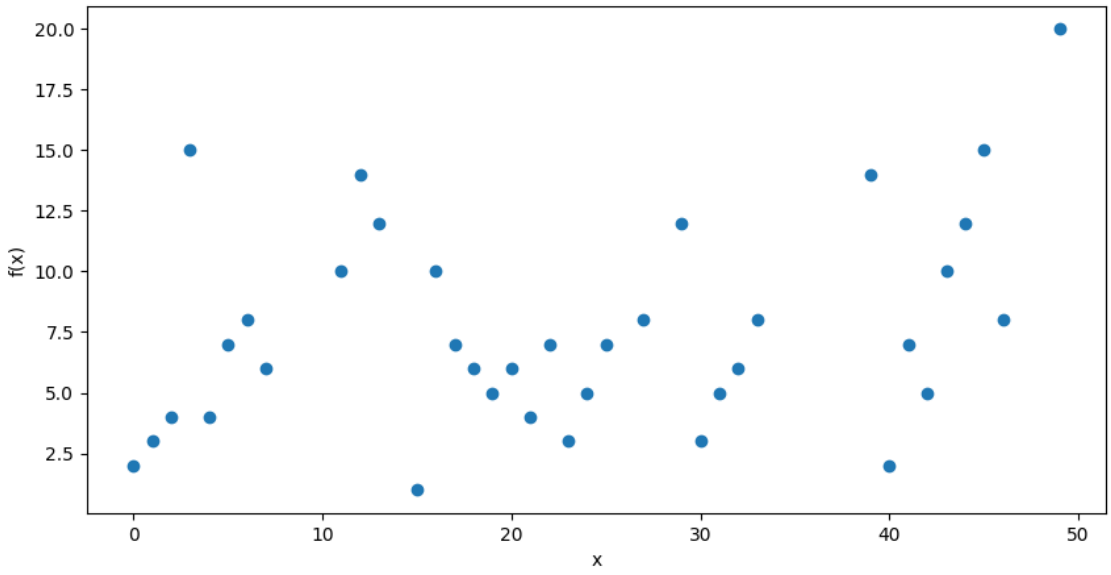
[17]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches
```

0.1 Daten erzeugen

[3]:
```
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,␣
 ↪19, 20, 21, 22, 23, 24, 25,
              26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,␣
 ↪42, 43, 44, 45, 46, 47, 48, 49])
y = np.array([2, 3, 4, 15, 4, 7, 8, 6, np.nan, np.nan, np.nan, 10, 14, 12, np.
 ↪nan, 1, 10, 7, 6, 5, 6, 4, 7, 3, 5, 7,
              np.nan, 8, np.nan, 12, 3, 5, 6, 8, np.nan, np.nan, np.nan, np.
 ↪nan, np.nan, 14, 2, 7, 5, 10, 12, 15, 8, np.nan, np.nan, 20])
```

0.2 Plot

[4]:
```
fig, ax = plt.subplots(figsize=(10, 5))

plt.scatter(x, y)

plt.xlabel('x')
plt.ylabel('f(x)')

plt.show()
```



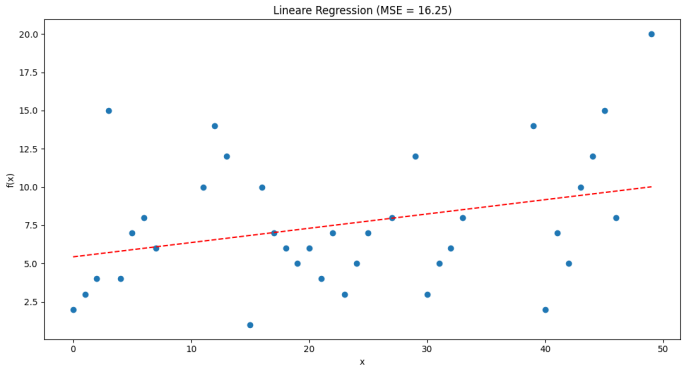
0.3 Aufgabe 1

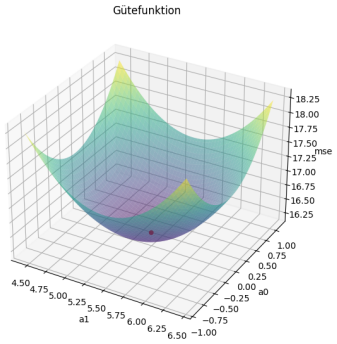
[5]:
```python
def polynomial_regression(x, y, degree):
    """
    Berechnet die Koeffizienten einer polynomialen Regression auf den gegebenen␣
 ↪Daten.

    Parameter:
    x (ndarray): Ein Array von X-Koordinaten.
    y (ndarray): Ein Array von Y-Koordinaten.
    degree (int): Der Grad des Polynoms.

    Rückgabe:
    a (ndarray): Ein Array von Koeffizienten des Polynoms.
    mse (float): Die mittlere quadratische Abweichung zum Polynom.
    """

    # Entferne fehlende Werte aus den Daten
    mask = ~np.isnan(y)
    x_clean = x[mask]
    y_clean = y[mask]

    # Erzeuge die Matrix Y aus X und den Potenzen von X bis zum Polynomgrad
    Y = np.ones_like(x_clean)
    for i in range(1, degree+1):
        Y = np.column_stack((Y, x_clean ** i))

    # Berechne die Koeffizienten des Polynoms
    Y_T = Y.T
    a = np.linalg.inv(Y_T.dot(Y)).dot(Y_T).dot(y_clean)

    # Berechne die mittlere quadratische Abweichung zum Polynom
    y_pred = Y.dot(a)
    mse = np.mean((y_clean - y_pred) ** 2)

    # Gib die Koeffizienten und die mittlere quadratische Abweichung aus
    return a, mse

def build_poly_func(coeffs):
    """
    Erstellt eine Polynomfunktion auf Grundlage der Koeffizienten.

    Parameter:
    coeffs (tuple): Ein Tuple von Koeffizienten in aufsteigender Reihenfolge.

    Rückgabe:
```

```
    func (function): Eine Funktion, die den Funktionswert des Polynoms für eine␣
 ↪gegebene x-Koordinate berechnet.
    """

    return sum([coeff * x ** i for i, coeff in enumerate(coeffs)])


def generate_cost_function(coeff, mse):
    """
    Generiert eine Gütefunktion für die lineare Regression.

    Parameter:
    coeff (ndarray): Ein Array von Koeffizienten der Regressionsgerade.
    mse (float): Die mittlere quadratische Abweichung zur Regressionsgerade.

    Rückgabe:
    a1 (ndarray): Ein Array von Werten für den Koeffizienten a1.
    a0 (ndarray): Ein Array von Werten für den Koeffizienten a0.
    mse_paraboloid (ndarray): Ein Array von Werten für die mittlere␣
 ↪quadratische Abweichung zur Regressionsgerade,
        berechnet als Funktion von a1 und a0.

    mse_paraboloid ist eine Approximation der Gütefunktion J(a_1, a_0) durch␣
 ↪eine parabolische Funktion.
    Die Gütefunktion J(a_1, a_0) ist eine Funktion von a_1 und a_0, die den␣
 ↪mittleren quadratischen Abstand
    zwischen der Regressionsgeraden und den tatsächlichen Datenpunkten angibt.

    Die Idee, eine parabolische Funktion zur Approximation der Gütefunktion zu␣
 ↪verwenden, beruht auf der Annahme,
    dass die Gütefunktion in der Nähe des Schätzwertes der Koeffizienten coeff␣
 ↪ein Minimum aufweist. Dieses Minimum wird durch die
    parabolische Funktion approximiert.

    Um die parabolische Funktion zu erzeugen, wird zuerst der Schätzwert der␣
 ↪Koeffizienten coeff als Schwerpunkt der Parabel verwendet.
    Dann wird um diesen Schwerpunkt eine Spanne von ±1 erzeugt und in 100␣
 ↪gleich große Schritte unterteilt, um die Werte für a_1 und a_0
    zu erhalten. Schließlich wird für jedes Paar von a_1 und a_0 der Wert von␣
 ↪J(a_1, a_0) durch die Formel
    mse_paraboloid = mse_0 + (a1 - a1_0) ** 2 + (a0 - a0_0) ** 2 berechnet.
    Dabei ist mse_0 die mittlere quadratische Abweichung zur Regressionsgerade␣
 ↪an der Stelle coeff, a1_0 und a0_0 sind die Schätzwerte
    für a_1 und a_0, und a1 und a0 sind die Werte, die um den Schwerpunkt der␣
 ↪Parabel erzeugt wurden.
    """
```

```
    a1_0 = coeff[0]
    a0_0 = coeff[1]
    mse_0 = mse

    a1 = np.linspace(a1_0 - 1, a1_0 + 1, 100)
    a0 = np.linspace(a0_0 - 1, a0_0 + 1, 100)
    a1, a0 = np.meshgrid(a1, a0)

    mse_paraboloid = mse_0 + (a1 - a1_0) ** 2 + (a0 - a0_0) ** 2

    return a1, a0, mse_paraboloid
```

[6]:
```
# Berechne die lineare Regression
coeff, mse = polynomial_regression(x=x, y=y, degree=1)
y_pred = build_poly_func(coeff)

# Erzeuge die parabolische Gütefunktion
a1, a0, mse_paraboloid = generate_cost_function(coeff, mse)

# Erzeuge den Subplot mit beiden Plots
fig = plt.figure(figsize=(20, 6))

# Plot der Daten und Regressionsgerade
ax1 = fig.add_subplot(121)
ax1.scatter(x, y)
ax1.plot(x, y_pred, 'r--')
ax1.set_xlabel('x')
ax1.set_ylabel('f(x)')
ax1.set_title('Lineare Regression (MSE = {:.2f})'.format(mse))

# 3D-Plot der Gütefunktion
ax2 = fig.add_subplot(122, projection='3d')
ax2.plot_surface(a1, a0, mse_paraboloid, cmap='viridis', alpha=0.5)
ax2.scatter(coeff[0], coeff[1], mse, color='red')
ax2.set_xlabel('a1')
ax2.set_ylabel('a0')
ax2.set_zlabel('mse')
ax2.set_title('Gütefunktion')

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





0.3.1 Kommentar zu den Ergebnissen

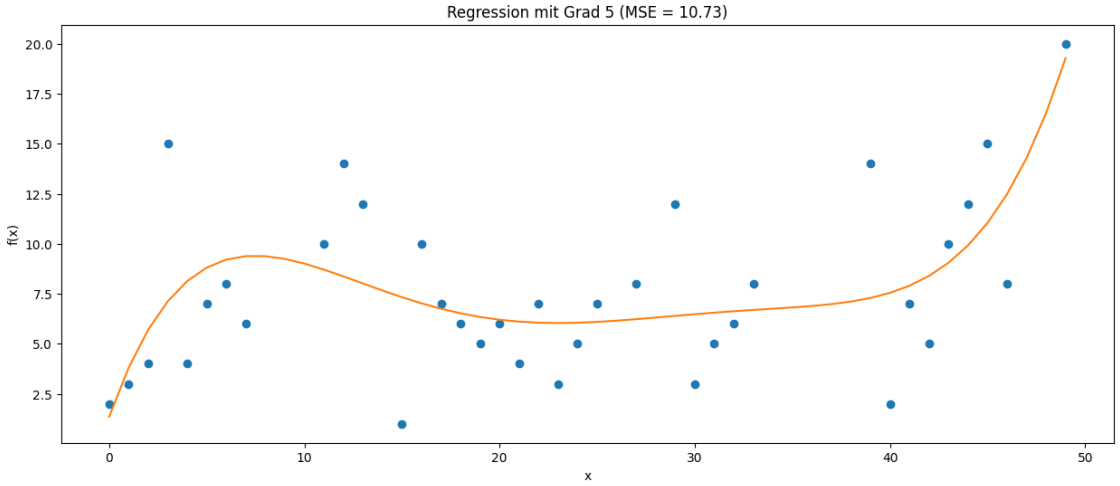
Aus den generierten Daten lässt sich kein klarer Funktionalszusammenhang erkennen. Sie scheinen zufällig und chaotisch zu sein. Daher ist es schwierig, ein lineares Modell zu finden, das die Beziehung zwischen x und y genau beschreibt.

Die durchgeführte lineare Regression liefert eine dennoch einen Funktionalzusammenhang und somit eine Regressionsgerade, die die Daten in beschreibt. Allerdings sind die Datenpunkte sehr weit von der Regressionsgerade entfernt, was darauf hinweist, dass die lineare Regression nicht das beste Modell für diese Daten ist.

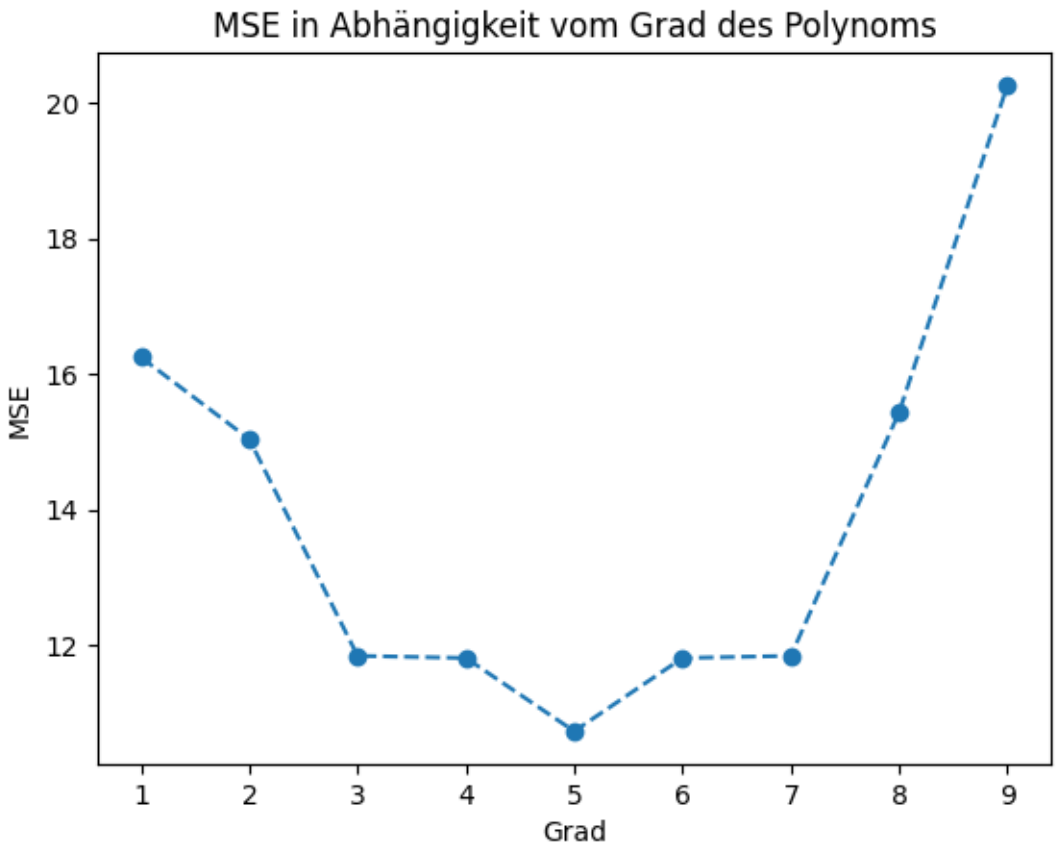
Der MSE ist relativ groß im Verhältnis zu dem Bereich der y-Werte, was ebenfalls darauf hindeutet, dass ein linearer Zusammenhang für diese Daten ungeeignet ist.

0.4 Aufgabe 2

[7]:
```
degree = 5

coef, mse = polynomial_regression(x=x, y=y, degree=degree)

poly_func = build_poly_func(coef)

plt.figure(figsize=(15, 6))
plt.plot(x, y, 'o', label='Daten')
plt.plot(x, poly_func, label='Polynom')
plt.xlabel('x')
plt.ylabel('f(x)')

plt.title('Regression mit Grad {} (MSE = {:.2f})'.format(degree, mse))
plt.show()
```



[8]:
```
degrees = np.arange(1, 10)
mses = []

for degree in degrees:
    coef, mse = polynomial_regression(x, y, degree)
    mses.append(mse)

plt.plot(degrees, mses, '--o')
plt.xlabel('Grad')
plt.ylabel('MSE')
plt.title('MSE in Abhängigkeit vom Grad des Polynoms')
plt.show()
```



0.4.1 Kommentar zu den Ergebnissen

Generell lässt sich sagen, dass mit zunehmendem Polynomgrad die Anpassungsgenauigkeit an die Datenpunkte steigt. Allerdings besteht auch die Gefahr von Overfitting, wenn das Polynom zu stark auf die Datenpunkte "überanpasst" wird. Es ist wichtig, den geeigneten Polynomgrad zu wählen, der eine gute Balance zwischen Anpassungsgenauigkeit und Modellkomplexität bietet.

Mit zunehmendem Grad des Polynoms wird eine höhere Flexibilität des Modells erreicht, um die Datenpunkte genauer anzupassen. Bei niedrigeren Polynomgraden, wie Grad 1 oder 2, passt sich das Polynom grob der Tendenz der Daten an, aber es gibt immer noch sichtbare Abweichungen.

Bei höheren Polynomgraden, wie Grad 3 oder 4, kann das Modell die Datenkurven genauer abbilden und passt sich den Datenpunkten besser an. Es werden detailliertere Muster erfasst und die Kurven folgen den Daten enger.

Allerdings gibt es einen Punkt, an dem die Zugabe höherer Polynomgrade keine wesentlichen Verbesserungen mehr bringt. Ab einem gewissen Grad (je nach Datenverteilung und Rauschniveau) führt die Erhöhung des Polynomgrads zu einer Überanpassung der Datenpunkte und zu einer komplexeren Modellstruktur. Dies kann zu einem erhöhten MSE führen und das Modell wird empfindlich gegenüber Rauschen und Ausreißern.

In unserem Fall scheint der Grad 5 das höchste Niveau zu sein, bei dem die Polynomregression eine

sichtbare Verbesserung bringt. Eine weitere Erhöhung des Polynomgrads würführt zu minimalen oder keinen Veränderungen an der Regressionskurve. Es ist wichtig, den geeigneten Polynomgrad zu wählen, der die Datenstruktur angemessen erfasst, ohne das Modell zu komplex zu machen.