1. Полиномиальная регрессия

Цель работы - смоделировать процесс измерения функциональной зависимости с шумом и изучить, как степень полиномиальной регрессии влияет на качество восстановления функции, включая проявления недообучение и переобучение.

Задания:

1. Генерация выборок с моделированием случайной ошибки при помощи заданных функций и их дальнейший анализ
2. Восстановление функциональной зависимости с помощью полиномиальной регрессии

Ссылка на исходный код выполненной работы:
https://github.com/F1ameX/Modern-Methods-of-Deep-Machine-Learning/blob/main/1_polynominal_regression/1_polynominal_regression.ipynb

Ход работы

Моделирование выборок для заданных функций с моделированием случайной ошибки измерения

Для проведения эксперимента для обеих функций f = ax^3 + bx^2 + cx + d и f = x * sin(2 * pi * x) были сгенерированы выборки данных.
Для полиномиальной функции коэффициенты a,b,c,d генерировались случайно и независимо по равномерному распределению на интервале [-3,3]:

```
rng = np.random.default_rng() # random generator
a, b, c, d = rng.uniform(low = -3.0, high = 3.0, size = 4) # random coefficient
```

Рисунок 1 - Генерация коэффициентов для полинома

Значения x были сгенерированы случайно равномерным распределением на интервале [-1, 1], а значения y получались в результате исполнения функций, указанных ранее, с добавлением случайной ошибки. Ниже приведен код генерации выборок:

```
for i in range(N):
    sample[i][0] = rng.uniform(low = -1.0, high = 1.0)

for func in range(2):
    clear = f(sample[:, 0], option = func)
    for distribution in range(2):
        for eps in epsilons:
            for i in range(N):

                if distribution == 0:
                    sample[i][1] = f(sample[i][0], option = func) + rng.uniform(low = -eps, high = eps)
                else:
                    sample[i][1] = f(sample[i][0], option = func) + np.clip(rng.normal(loc = 0.0, scale = eps / 3), -eps, eps)
```

Рисунок 2 - Генерация данных

Случайная ошибка генерировалась из интервала [-eps, eps] случайно равномерным или нормальным распределением. При этом сами значения eps перебирались из множества [0.001, 0.1, 1, 1.5]. Для нормально распределенной ошибки использовались параметры мат. Ожидания m = 0 и дисперсии sigma =epsilon / 3. Таким образом, при переборе различных значений epsilon одновременно изменялась и дисперсия нормального распределения, что позволяет рассмотреть разные уровни “нормального” зашумления. При формировании случайной ошибки нормальным распределением дополнительно использовалась функция clip, для того, чтобы значения лежали в заданном диапазоне. Итоговая функция формирования для нормального распределения выглядит следующим образом:

$$z_i \sim \mathcal{N}\left(0, \left(\frac{\varepsilon_0}{3}\right)^2\right), \qquad \varepsilon_i = \mathrm{clip}(z_i, -\varepsilon_0, +\varepsilon_0),$$

$$\mathrm{clip}(z, a, b) = \min\{\max\{z, a\}, b\}.$$

Рисунок 3 - Формула генерации случайной ошибки нормальным распределением

После генерации всех выборок были построены графики, отражающие изначальную функцию и функцию с добавлением случайной ошибки. Рассмотрим ниже некоторые полученные примеры для полиномиальной функции:

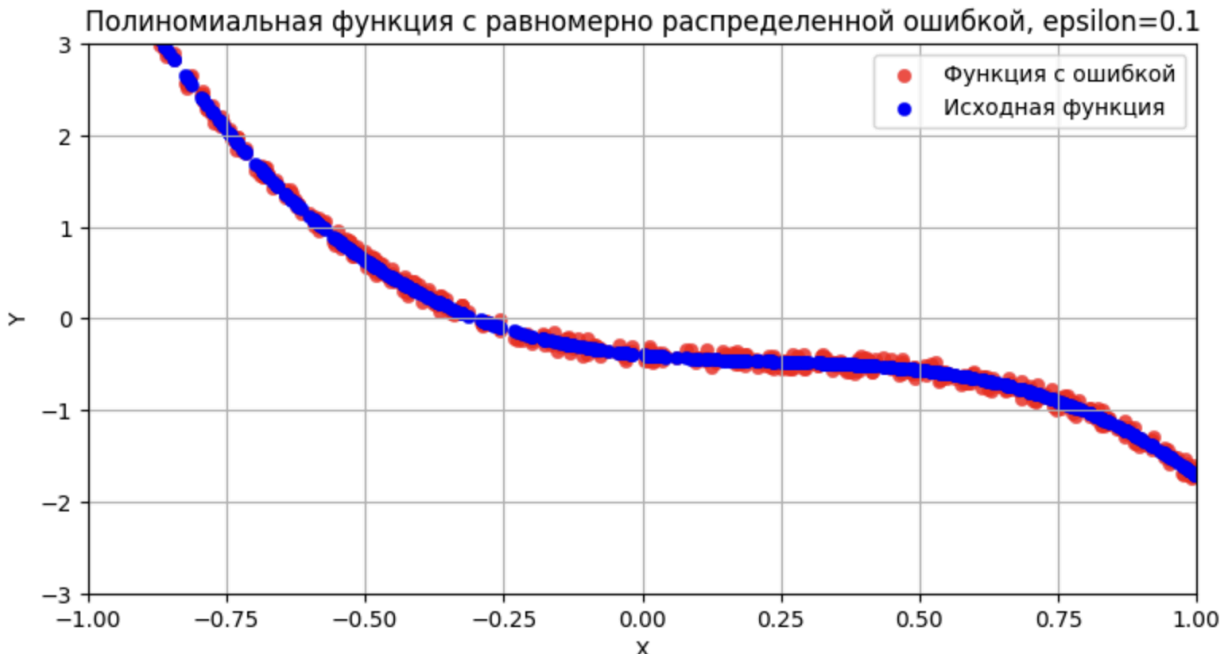

Рисунок 4 - Полиномиальная функция с равномерно распределенной ошибкой (eps = 0.1)

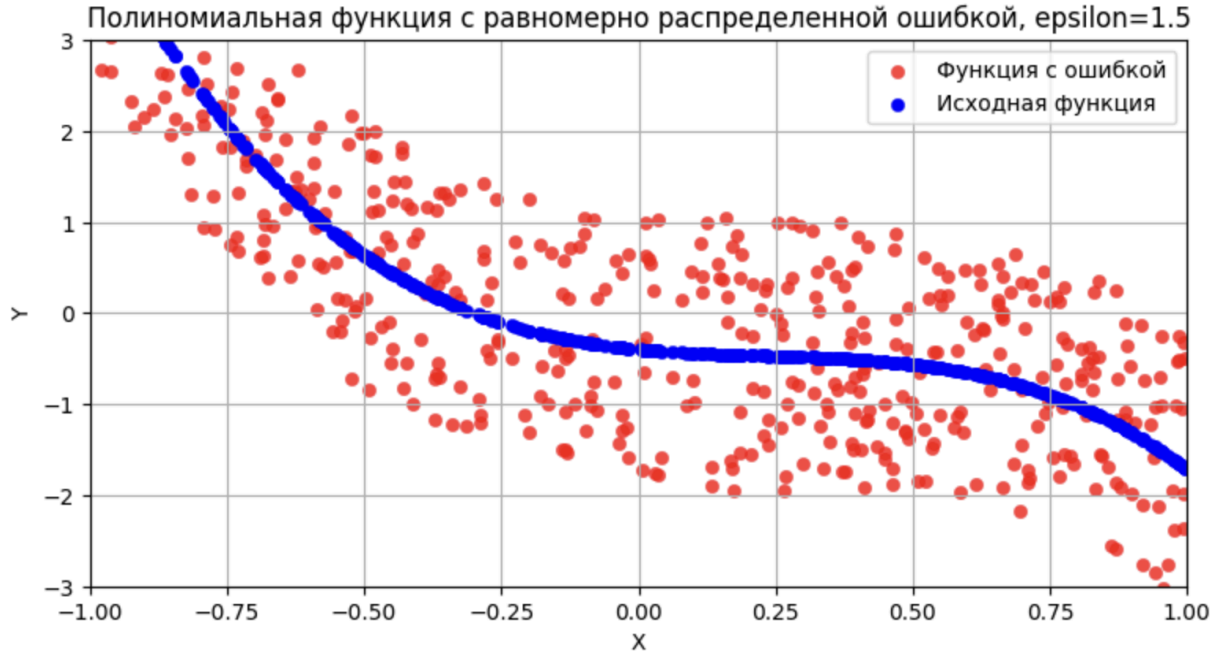

Рисунок 5 - Полиномиальная функция с равномерно распределенной ошибкой (eps = 1.5)

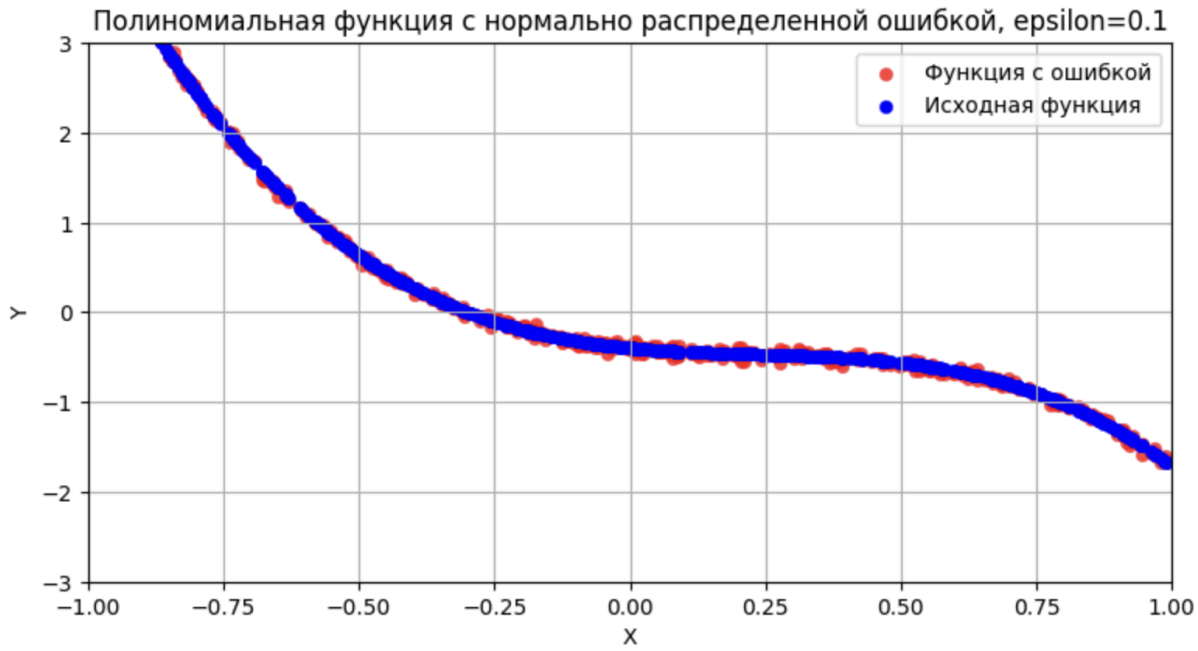

Рисунок 6 - Полиномиальная функция с нормально распределенной ошибкой (eps = 0.1)

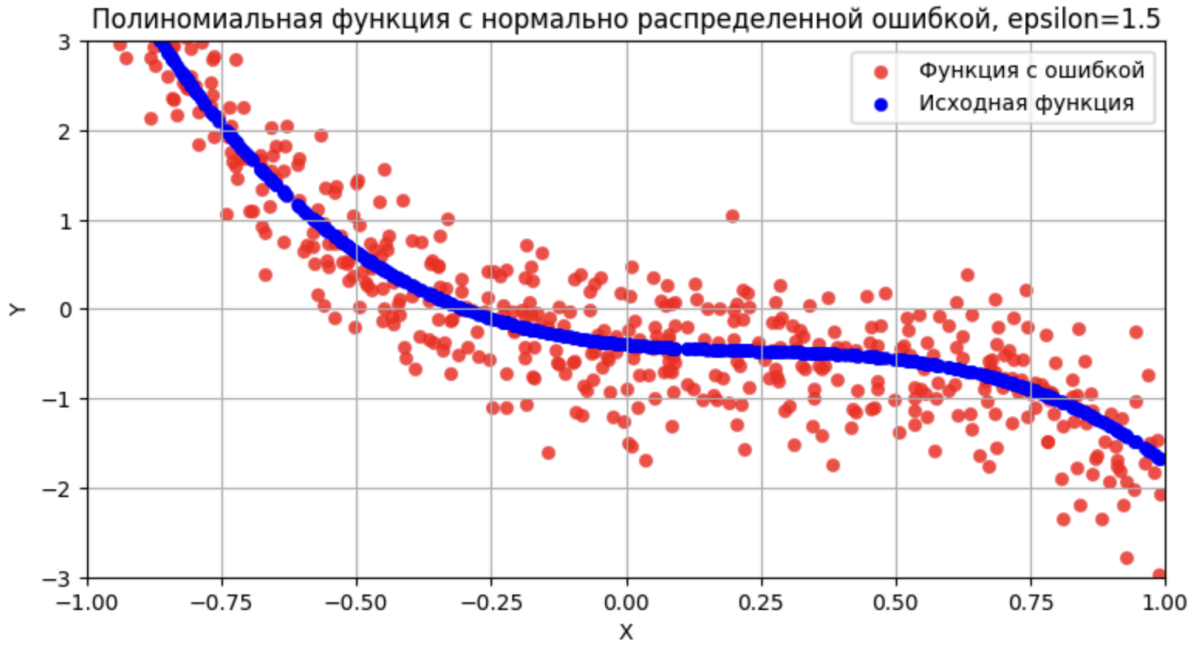

Рисунок 7 - Полиномиальная функция с нормально распределенной ошибкой (eps = 1.5)

По рисункам 4–7 выше видно, что характер зашумления существенно зависит от распределения ошибки, а также её амплитуды epsilon. При малых значениях epsilon различия между равномерной и нормальной ошибкой

минимальны: обе модели дают выборку, близкую к исходной функции, а отклонения выглядят как небольшие локальные флуктуации вокруг истинной зависимости. При большем же значении epsilon наблюдается сильное размытие точек относительно графика функции, из-за чего визуально хуже считывается исходная форма зависимости и возрастает неопределенность в восстановлении параметров модели.

Дополнительно заметно, что нормальная ошибка приводит к более неравномерной структуре рассеяния: основная масса точек концентрируется ближе к графику функции, но встречаются редкие, заметно удалённые отклонения (выбросы), связанные с “хвостами” распределения. В случае равномерной ошибки разброс более однородный по всему диапазону, а значения строго ограничены сверху и снизу, поэтому точки формируют относительно равномерную “полосу” вокруг исходной кривой без ярко выраженных одиночных выбросов.

Теперь же рассмотрим полученные примеры для синусоидальной функции:

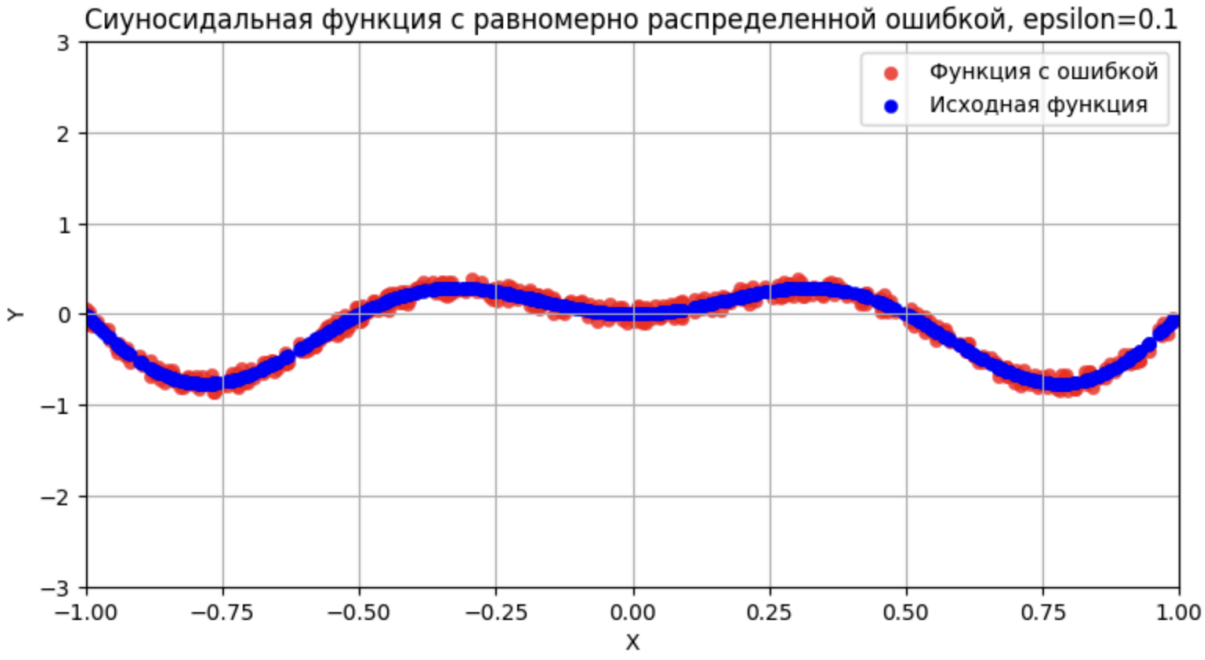

Рисунок 8 - Синусоидальная функция с равномерно распределенной ошибкой (eps = 0.1)

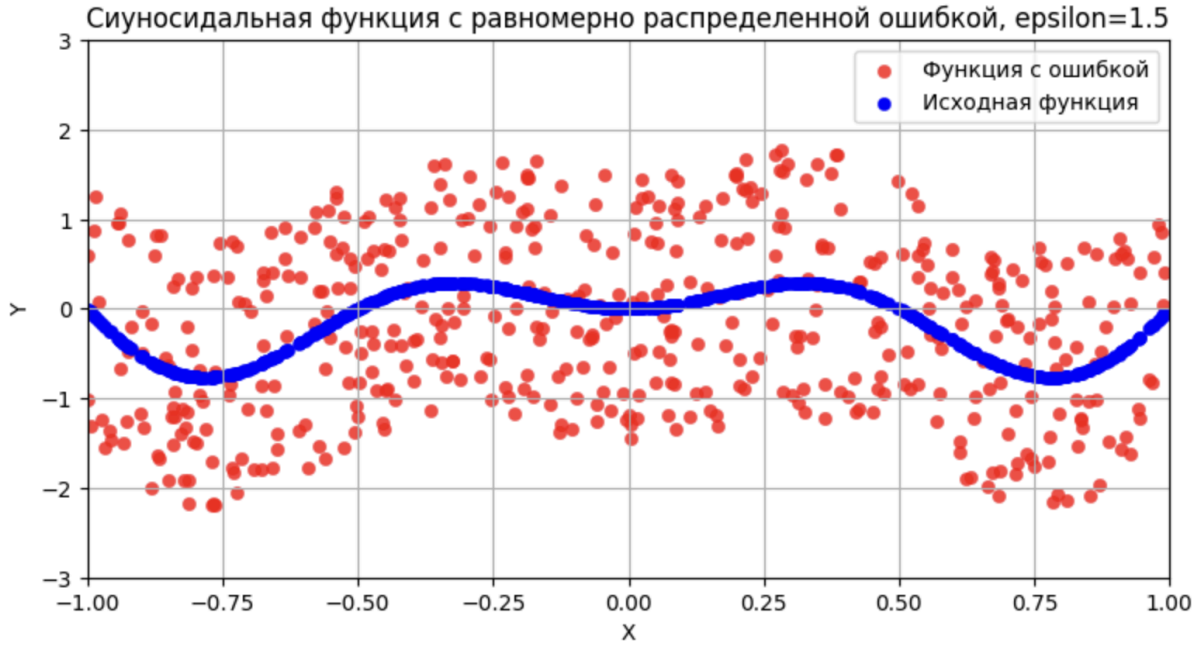

Рисунок 9 - Синусоидальная функция с равномерно распределенной ошибкой (eps = 1.5)

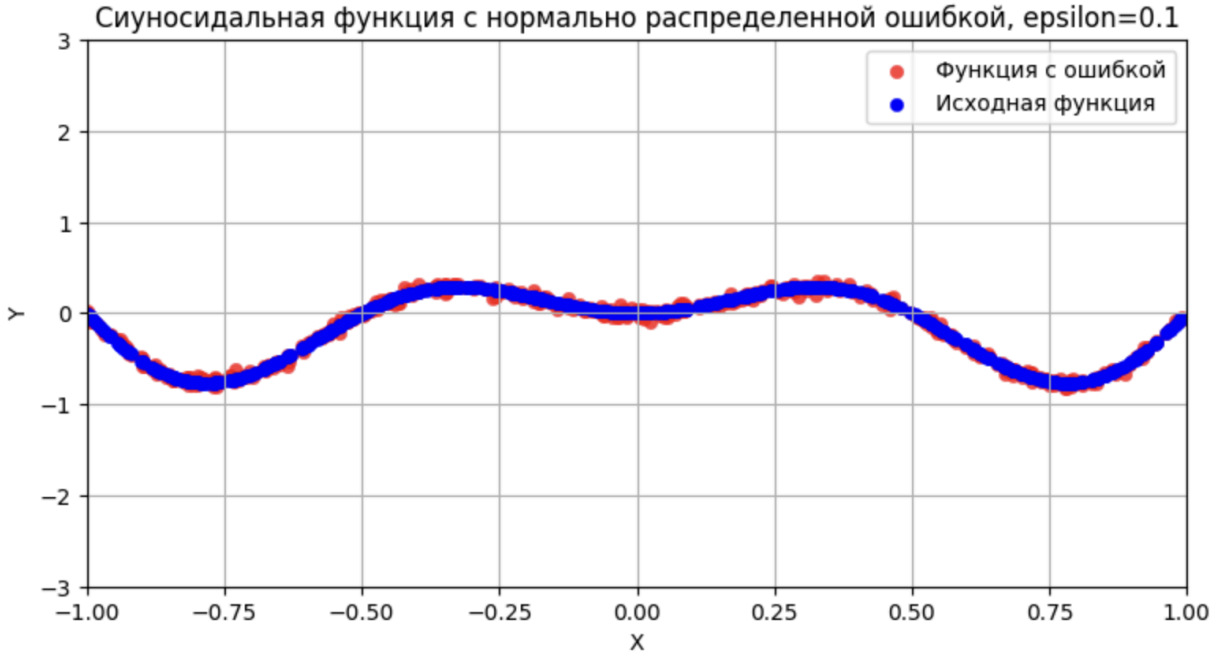

Рисунок 10 - Синусоидальная функция с нормально распределенной ошибкой (eps = 0.1)

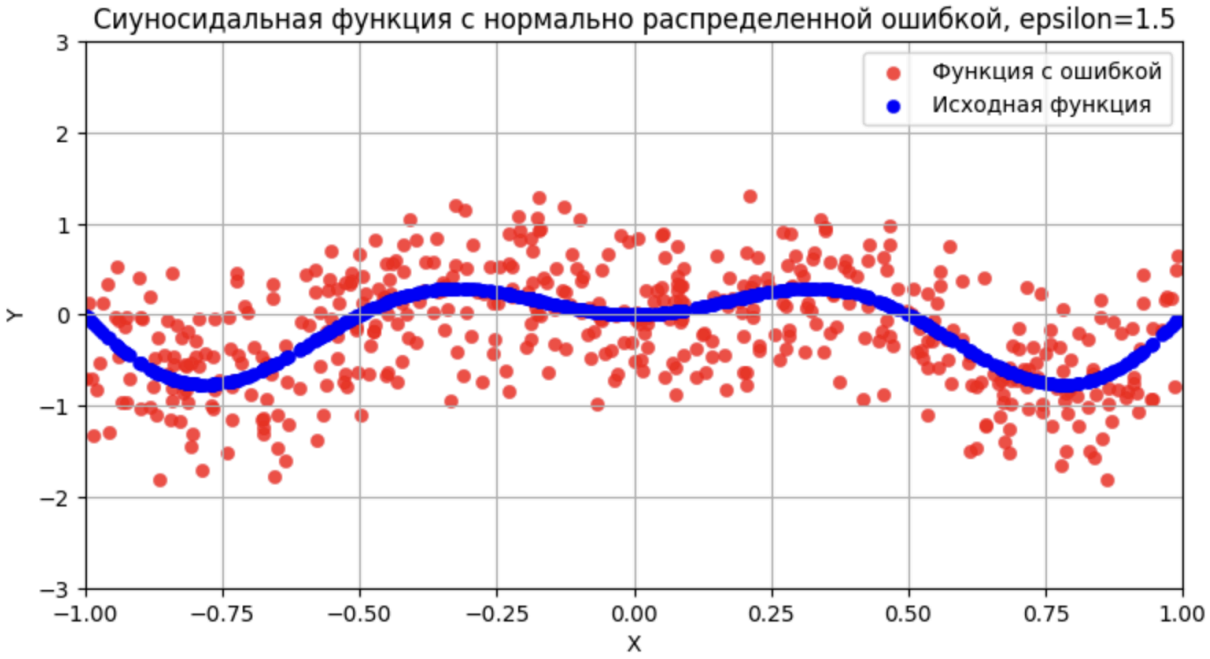

Рисунок 11 - Синусоидальная функция с нормально распределенной ошибкой (eps = 1.5)

По рисункам 8-11 видно, что для синусоидальной функции влияние распределения ошибки и параметра epsilon проявляется аналогично случаю полинома, однако визуально заметнее из-за гладкой волнообразной формы исходной зависимости. При epsilon = 0.1 (рис. 8 и 10) точки с шумом практически повторяют исходную функцию: отклонения небольшие, структура “волн” сохраняется, а влияние шума носит локальный характер. Различия между равномерным и нормальным шумом при таком уровне амплитуды минимальны - обе модели ошибки дают близкую к функции выборку.

При увеличении амплитуды до epsilon = 1.5 (рис. 9 и 11) наблюдается существенное “размытие” данных: точки распределяются в широком вертикальном диапазоне, и исходная функциональная зависимость частично маскируется шумом. При равномерно распределенной ошибке (рис. 9) отклонения ограничены сверху и снизу, поэтому область зашумления имеет почти равномерную “полосу” вокруг исходной кривой. В случае нормально распределенной ошибки с ограничением (рис. 11) основная масса точек концентрируется ближе к исходной функции, однако появляются редкие наблюдения, лежащие значительно дальше (вблизи границ диапазона), что

визуально воспринимается как более выраженные “выбросы”. Это связано с хвостами нормального распределения и последующим ограничением значений ошибки по диапазону [-epsilon, epsilon].

Восстановление функциональной зависимости с помощью полиномиальной регрессии

В данной части эксперимента выполняется восстановление функциональной зависимости заданной функции по заданной зашумленной выборке методом полиномиальной регрессии. Модель аппроксимации задается полиномом степени m. Ниже приведен алгоритм аппроксимации:

$$\hat{y}(x) = \sum_{k=0}^{m} w_k x^k,$$

где m — степень полинома, а w_k — параметры модели.

Для обучения вводится вектор полиномиальных признаков:

$$\varphi(x_i) = \left(1,\, x_i,\, x_i^2,\, \ldots,\, x_i^m\right)^\top,$$

и формируется дизайн-матрица (матрица Вандермонда) $\Phi \in \mathbb{R}^{N \times (m+1)}$:

$$\Phi = \begin{pmatrix} \varphi(x_1)^\top \\ \varphi(x_2)^\top \\ \vdots \\ \varphi(x_N)^\top \end{pmatrix}.$$

Тогда вектор предсказаний на обучающей выборке записывается как $\hat{y} = \Phi w$. Параметры находятся из задачи метода наименьших квадратов:

$$w^* = \arg\min_w \|\Phi w - y\|_2^2.$$

Качество восстановления оценивается по среднеквадратичной ошибке:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}(x_i)\right)^2.$$

Рисунок 12 - Алгоритм аппроксимации при помощи полиномиальной регрессии

Для проведения эксперимента был реализован класс полиномиальной регрессии:

```
class PolynominalRegression():
    def __init__(self, degree : int) -> None:
        self.degree = degree
        self.weights = None

    def _design_matrix(self, X: np.ndarray) -> np.ndarray:
        X = X.reshape(-1, 1)
        return np.hstack([X**i for i in range(self.degree + 1)])

    def _loss(self, y_true: np.ndarray, y_pred: np.ndarray) -> float:
        y_true = y_true.reshape(-1, 1)
        y_pred = y_pred.reshape(-1, 1)
        return np.mean((y_true - y_pred) ** 2)

    def fit(self, X: np.ndarray, y: np.ndarray, ridge_alpha: float = 0.0):
        X_poly = self._design_matrix(X)
        y = y.reshape(-1, 1)

        if ridge_alpha == 0.0:
            self.weights, *_ = np.linalg.lstsq(X_poly, y, rcond=None)
        else:
            I = np.eye(X_poly.shape[1])
            self.weights = np.linalg.solve(X_poly.T @ X_poly + ridge_alpha * I, X_poly.T @ y)

        return self


    def predict(self, X: np.array) -> np.array:
        if self.weights is not None:
            X_poly = self._design_matrix(X)
            return (X_poly @ self.weights).ravel()
```

Рисунок 13 - Код реализации полиномиальной регрессии

После этого генерировалась новая выборка с добавлением случайного шума к значениям целевой переменной, аналогичным путем, что и ранее (рис. 2). Сгенерированная выборка разбивалась на обучающую и тестовую части в отношении 60 на 40 соответственно.

```
def split_data(X: np.array, y: np.array, ratio : float = 0.33, random_state : int = 42):
    rng = np.random.default_rng(seed = random_state)
    idx = np.arange(X.shape[0])
    rng.shuffle(idx)
    left_share = int((1 - ratio) * X.shape[0])

    X_left = X[idx[: left_share]]
    y_left = y[idx[: left_share]]
    X_right = X[idx[left_share: ]]
    y_right = y[idx[left_share: ]]

    return X_left, y_left, X_right, y_right
```

Рисунок 14 - Код реализации функции для разбиения на выборки

Далее для набора степеней m обучалась полиномиальная регрессия, после чего вычислялись значения. По характеру зависимости ошибок от m и по виду аппроксимирующей кривой выбирались примеры, соответствующие недообучению, удовлетворительному обобщению и переобучению. Качество оценивалось по среднеквадратичной ошибке (MSE) на обучающей и тестовой выборках.

```
new_sample = np.empty((N, 2), dtype = float)

for i in range(N):
    new_sample[i][0] = rng.uniform(low = -1.0, high = 1.0)
    new_sample[i][1] = f(new_sample[i][0], option = 1)

X_train, y_train, X_test, y_test = split_data(new_sample[:, 0], new_sample[:, 1], ratio = 0.6)

results = []

for m in range(21):
    model = PolynominalRegression(degree = m)
    model.fit(X_train, y_train)

    MSE_train = model._loss(y_train, model.predict(X_train))
    MSE_test = model._loss(y_test, model.predict(X_test))

    results.append({'Степень полинома' : m, 'MSE на обучающей выборке' : MSE_train, 'MSE на тестовой выборке' : MSE_test})
```

Рисунок 15 - Код реализации обучения полиномиальных регрессий на разных полиномах

Полученные данные были структурированы в таблицу, которая была проанализирована (см. рис. 16)

Исходя из полученных результатов (рис. 16), были выбраны степени полиномов, демонстрирующие недообучение, удовлетворительное обобщение и переобучение. В качестве примера недообучения выбрана степень m = 1, так как при такой модели аппроксимация слишком проста и не способна описать форму зависимости: ошибки на обучающей и тестовой выборках остаются высокими MSE_train = 0.320243, MSE_test = 0.148174.
В качестве примера нормального обучения выбрана степень m = 8, поскольку при ней достигается минимальная ошибка на тестовой выборке MSE_test = 0.019126 при сравнительно низкой ошибке на обучающей выборке MSE_train = 0.188506, что указывает на хорошую обобщающую способность модели.
Наконец, в качестве примера переобучения выбрана высокая степень m = 20: обучающая ошибка продолжает уменьшаться MSE_train = 0.155764, однако ошибка на тестовой выборке резко возрастает MSE_test = 0.878461, что

соответствует подстройке модели под шум обучающих данных и ухудшению качества обобщения.

Степень полинома	MSE на обучающей выборке	MSE на тестовой выборке
0	0.335314	0.142364
1	0.320243	0.148174
2	0.264189	0.065296
3	0.263264	0.065049
4	0.243572	0.047457
5	0.243084	0.049642
6	0.189716	0.019467
7	0.188627	0.021222
8	0.188506	0.019126
9	0.186587	0.024727
10	0.180780	0.034902
11	0.172431	0.059652
12	0.169302	0.051214
13	0.166313	0.076798
14	0.166132	0.070975
15	0.160681	0.150101
16	0.160681	0.150224
17	0.158298	0.305172
18	0.158266	0.278512
19	0.158100	0.362687
20	0.155764	0.878461

Рисунок 16 - Таблица результатов полиномов

Рассмотрим степени полученных полиномов:

1. Полином степени 1 демонстрирует явное недообучение (см. рис. 17)
2. Полином степени 8 демонстрирует нормально обучение (см. рис. 18)
3. Полином степени 20 демонстрирует явное переобучение (см рис. 19)

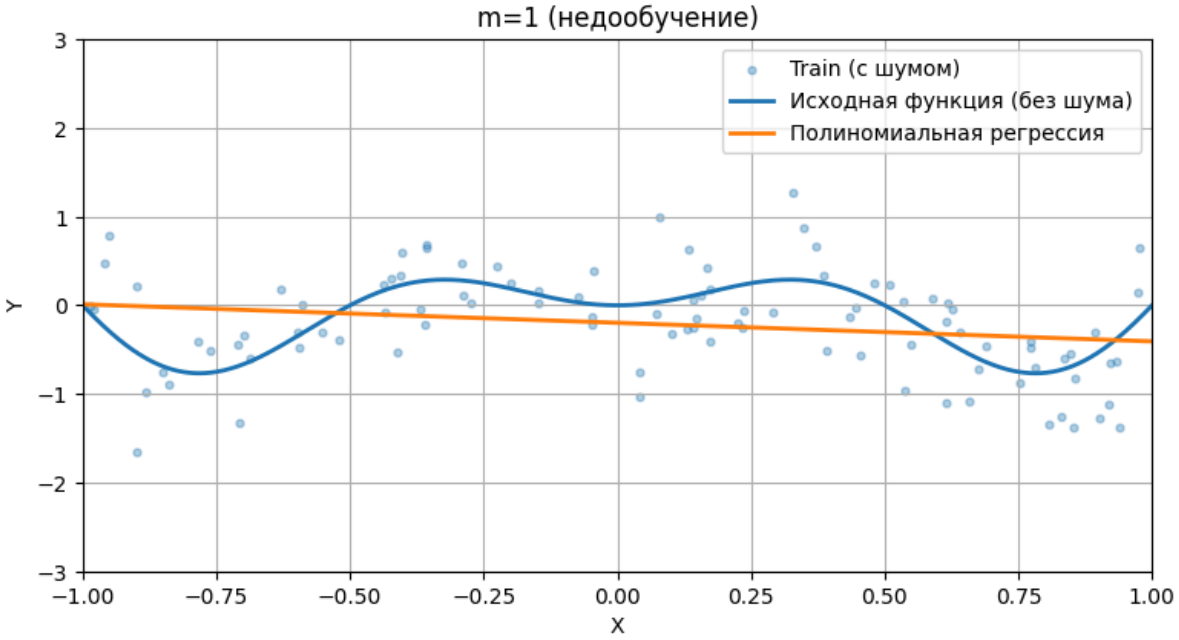

Рисунок 17 - Полиномиальная регрессия со степенью полинома 1 (недообучение)

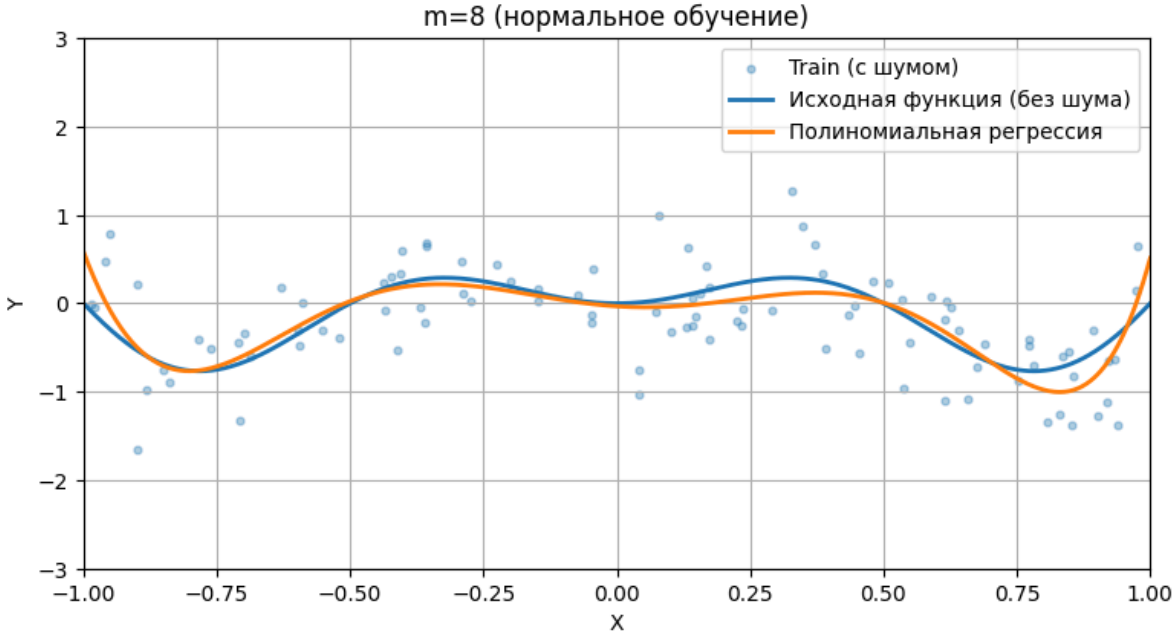

Рисунок 18 - Полиномиальная регрессия со степенью полинома 8 (нормальное обучение)

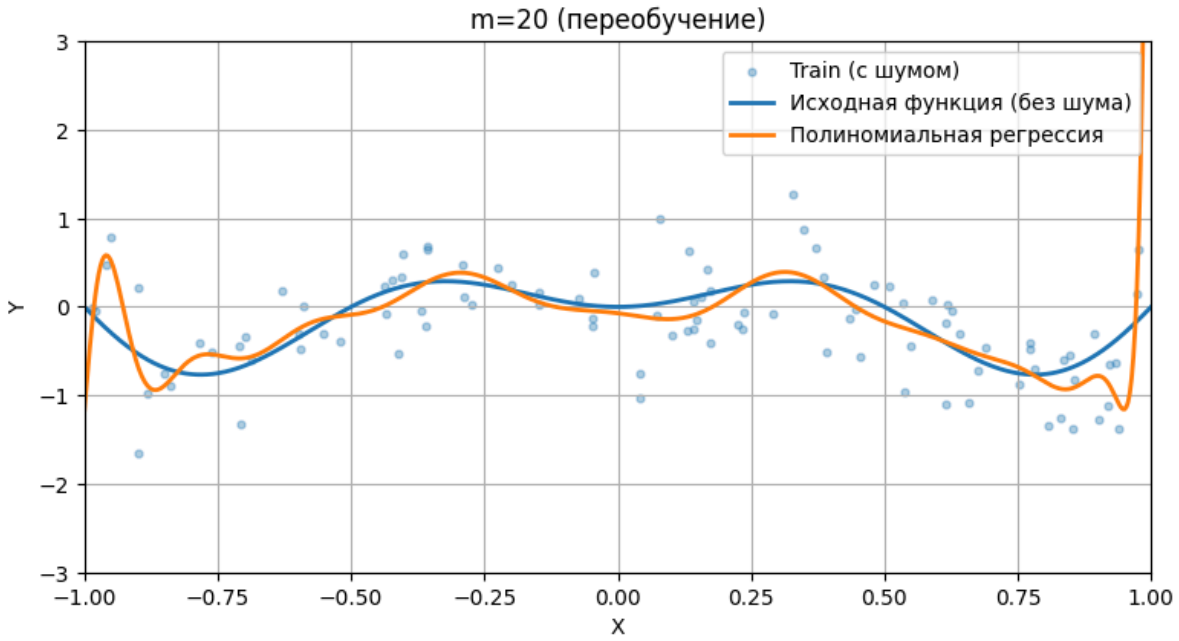

Рисунок 19 - Полиномиальная регрессия со степенью полинома 20 (переобучение)

По результатам эксперимента построены графики аппроксимации исходной функции полиномиальной регрессией при разных степенях полинома. Наблюдается типичное поведение модели при изменении сложности:

Низкая степень (недообучение): аппроксимирующая кривая имеет слишком простую форму и не способна воспроизвести характерные особенности исходной зависимости. В результате модель дает большую ошибку как на обучающей, так и на тестовой выборке. Визуально это проявляется в заметном расхождении кривой регрессии с исходной функцией на всём диапазоне x.

Умеренная степень (удовлетворительная аппроксимация): при увеличении степени полинома модель начинает корректно описывать форму зависимости, хорошо повторяя изгибы исходной функции. Ошибки на обучающей и тестовой выборках становятся малыми и сопоставимыми, что указывает на хорошую обобщающую способность модели.

Высокая степень (переобучение): при дальнейшем росте степени полинома модель начинает подстраиваться под шум обучающих данных. В результате

ошибка на обучающей выборке продолжает уменьшаться, однако ошибка на тестовой выборке заметно возрастает. Визуально это выражается в появлении лишних колебаний аппроксимирующей кривой и ухудшении совпадения с исходной функцией, что свидетельствует о снижении обобщающей способности модели.

Выводы

В ходе выполнения задания была рассмотрена задача восстановления функциональной зависимости по дискретной выборке с использованием полиномиальной регрессии. Была реализована модель полиномиальной регрессии на основе метода наименьших квадратов: сформирован вектор полиномиальных признаков и дизайн-матрица (матрица Вандермонда), получены параметры модели, а также реализована функция предсказания. Для визуального анализа качества восстановления построены графики исходной зависимости и аппроксимации.

Также был проведён эксперимент по исследованию влияния степени полинома на качество аппроксимации при наличии шума в обучающих данных. Данные разделялись на обучающую и тестовую выборки, а качество оценивалось по среднеквадратичной ошибке (MSE) на обеих выборках. Результаты показали, что степень полинома является ключевым параметром, определяющим сложность модели: при малых значениях степени наблюдается недообучение, при умеренных — достигается минимальная ошибка на тестовой выборке (наилучшее обобщение), а при больших — возникает переобучение, когда обучающая ошибка продолжает снижаться, но тестовая ошибка возрастает из-за подстройки под шум.