Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 804 «Теория вероятностей и компьютерное моделирование»

Курсовая работа на тему:
«Метод наименьших квадратов»
Вариант № 25

Выполнил(а) студент группы М8О-305Б-21
Титеев Рамиль Маратович
Преподаватель: Игнатов Алексей Николаевич
Оценка ___________________________

Дата ___________________________

Москва, 2023

Описание работы

Модель полезного сигнала имеет вид:

$$y(x) = \theta_0 + \theta_1 x + ... + \theta_m x^m \qquad (1)$$

Рассматривается модель наблюдений

$$y_k = \theta_0 + \theta_1 x_k + ... + \theta_m x_k^m + \varepsilon_k, \qquad k = \overline{1, n} \qquad (2)$$

где $\varepsilon_1, \varepsilon_2, ..., \varepsilon_n$ - независимые и одинаково распределенные случайные величины

Моделирование данных

Смоделировать два набора наблюдений на основе модели (2) для следующих случаев:

1 случай	**2 случай**
$m = 3, \varepsilon_k \sim N(0, \sigma^2)$	$m = 2, \varepsilon_k \sim R(-3\sigma, 3\sigma)$
$x_k = -4 + k\frac{8}{n}, \qquad k = \overline{1, n}, \qquad n = 40$	

Параметры задания определяются номером варианта:

Вариант 25

$$\theta_0 = (-1)^{N+1}\frac{N}{2} = (-1)^{25+1}\frac{25}{2} = 12.5$$

$$\theta_1 = -1$$

$$\theta_2 = -2$$

$$\theta_3 = 0.16$$

$$\sigma^2 = 1.6$$

Задание

1. Подобрать порядок многочлена $\hat{m} \geq 1$ в модели (1), используя критерий Фишера на уровне значимости 0.05, и вычислить оценки неизвестных параметров $(\theta_0, \ldots, \theta_{\hat{m}})$ методом наименьших квадратов.

2. В предположении нормальности ошибок построить доверительные интервалы уровней надежности $\alpha_1 = 0.95$ и $\alpha_2 = 0.99$ для параметров $(\theta_0, \ldots, \theta_{\hat{m}})$.

3. В предположении нормальности ошибок построить доверительные интервалы уровней надежности $\alpha_1 = 0.95$ и $\alpha_2 = 0.99$ для полезного сигнала (1).

4. Представить графически
   - истинный полезный сигнал
   - набор наблюдений
   - оценку полезного сигнала, полученную в шаге 1
   - доверительные интервалы полезного сигнала, полученные в шаге 3

5. По остаткам регрессии построить оценку плотности распределения случайной ошибки наблюдения в виде гистограммы.

6. В предположении нормальности ошибок вычислить оценку макимального правдоподобия дисперсии σ^2 случайной ошибки

7. По остаткам регрессии с помощью χ^2-критерия Пирсона на уровне значимости 0.05 проверить гипотезу о том, что закон распределения ошибки наблюдения является нормальным.

Выполнение работы

Импортируем класс модели

Класс модели описан в приложении №1

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, t, chi2
import statsmodels.api as sm
from sympy import symbols, Matrix, sqrt
from model import Model
```

Определения

При выполнении курсовой работы используется понятие `p-value`:

`p-value` — вероятность получить для данной модели распределения значений случайной величины такое же или более экстремальное значение статистики, по сравнению с ранее наблюдаемым, при условии, что нулевая гипотеза верна.

Случай 1

$m = 3, \varepsilon_k \sim N(0, \sigma^2)$

Инициализируем модель с заданными параметрами для первого случая

[2]:
```
model_1 = Model(m=3, theta=[12.5, -1.0, -2.0, 0.16], sigma=1.6, normal_eps=True)
```

[3]:
```
print("Вектор ошибок:\n")
print(model_1.eps)
```

```
Вектор ошибок:

[ 0.55269194  1.11873939 -0.59341989  0.05396172  0.07991609 -0.54703861
 -0.78076125  2.30413716 -0.42471399 -2.91245739  1.31340426 -2.04715994
 -0.77327656  2.1862464  -0.97246783  0.31204696  0.54877187  1.85215906
  0.95688657 -0.10385325  1.62111542  0.09520872  0.97819124 -0.63494703
 -1.73841928 -1.69063207 -0.38013809  0.7746026  -0.3511486   1.08807013
 -0.7999764  -0.58594985 -2.60135166 -1.06792011  1.31047878  2.91486564
 -0.19640127 -1.00732849  0.90965822  0.94305845]
```

[4]:
```
print("список X, на которых строятся наблюдения:\n")
print(model_1.x_k)
```

```
список X, на которых строятся наблюдения:

[-3.8 -3.6 -3.4 -3.2 -3.  -2.8 -2.6 -2.4 -2.2 -2.  -1.8 -1.6 -1.4 -1.2
 -1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8  1.   1.2  1.4  1.6
  1.8  2.   2.2  2.4  2.6  2.8  3.   3.2  3.4  3.6  3.8  4. ]
```

[5]:
```python
print("Полученные наблюдения:\n")
print(model_1.y_obs)
```

```
Полученные наблюдения:

[-20.80682806 -16.16622061 -14.10205989  -9.96891828  -6.74008391
  -4.43935861  -2.01292125   3.47229716   2.89160601   2.30754261
   8.20028426   6.27748006   8.76768344  12.7297664   10.36753217
  12.25012696  12.89421187  14.42191906  13.57560657  12.39614675
  13.84239542  11.88544872  12.19275124   9.86697297   7.92158072
   7.00584793   7.23890191   7.2099626    4.8019714    4.86807013
   1.5237036    0.20589015  -3.40919166  -3.53560011  -2.86952122
  -3.02225436  -7.92776127 -10.56236849 -10.49082178 -12.31694155]
```

Пункт 1. Вычисление порядка многочлена для модели, нахождение оценки параметров Проверка значимости коэффициентов регрессии означает проверку основной гипотезы об их значимом отличии от нуля.

Основная гипотеза состоит в предположении о незначимости коэффициента модели регрессии, т. е.

$$H_0 : \hat{\theta}_{\hat{m}} = 0$$

Обратная или конкурирующая гипотеза состоит в предположении о значимости коэффициентов модели регрессии, т. е.

$$H_1 : \hat{\theta}_{\hat{m}} \neq 0$$

Для проверки гипотезы на уровне значимости α используем статистический критерий $T_i = \frac{\hat{\theta}_i}{\mu_i}$, где $\hat{\theta}_i$ – оценка параметра модели, а μ_i – ее стандартная ошибка

$$\mu_i = \sqrt{\hat{\sigma}^2 * (X^T X)^{-1}_{ii}}$$

$\hat{\sigma}^2$ - оценка дисперсии ошибок

$$\hat{\sigma}^2 = \frac{(y_i - \hat{y}_i)^2}{n - \hat{m} - 1}$$

где n - кол-во наблюдений, $\hat{m}$ - степень многочлена

Для нахождения степени многочлена будем итеративно перебирать степень $\hat{m}$ пока не найдем такую, что при $\hat{m} + 1$ выполняется H_0.

Функция для итеративного T-теста описана в приложении №2

[6]:
```python
from model import iterative_t_test
```

[7]:
```python
m_hat = 1
print('================================================================================')
print('m_estimate = ', m_hat)
while iterative_t_test(model_1.x_k, model_1.phi(model_1.x_k)+model_1.eps, m_hat):
    m_hat+=1
```

```
     ␣
    ↪print('==========================================================================')
    print('m_estimate = ', m_hat)
m_hat -= 1
```

```
==============================================================================
m_estimate =  1
theta_1 не является статистически значимым.
```

Так как при $\hat{m} = 1$: $\hat{\theta}_1$ не является статистически значимым $\Rightarrow$ полагаем, что $\hat{m} = 0$

Расчитаем коэфициенты:

[8]:
```
model_1.fit(m_hat)
for i in range(len(model_1.params)):
    print(f"theta_{i} = {model_1.params[i]}")
```

```
theta_0 = 2.0186212264999988
```

Воспользуемся специализированной статистической библиотекой на `python` и проверим результы полученные в библиотеке с теми, которые получили на прошлом этапе

[9]:
```
X = np.vander(model_1.x_k, m_hat+1, increasing=True)
model = sm.OLS(model_1.y_obs, sm.add_constant(X))
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                       nan
Date:                Mon, 04 Dec 2023   Prob (F-statistic):                nan
Time:                        15:27:37   Log-Likelihood:                -146.54
No. Observations:                  40   AIC:                             295.1
Df Residuals:                      39   BIC:                             296.8
Df Model:                           0
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.0186      1.511      1.336      0.189      -1.038       5.075
==============================================================================
Omnibus:                        3.488   Durbin-Watson:                   0.071
Prob(Omnibus):                  0.175   Jarque-Bera (JB):                2.981
Skew:                          -0.569   Prob(JB):                        0.225
Kurtosis:                       2.297   Cond. No.                         1.00
==============================================================================

Notes:
```

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Пункт 2. Доверительные интервалы для параметров Для рассчета доверительных интервалов для параметров воспользуемся формулой:

$$\theta_i \in (\hat{\theta}_i - T_{crit} * \mu_i; \hat{\theta}_i + T_{crit} * \mu_i)$$

где μ_i - это стандартная ошибка параметра

$$\mu_i = \sqrt{\hat{\sigma}^2 * (X^T X)^{-1}_{ii}}$$

$\hat{\sigma}^2$ - оценка дисперсии ошибок

$$\hat{\sigma}^2 = \frac{(y_i - \hat{y}_i)^2}{n - \hat{m} - 1}$$

где n - кол-во наблюдений, $\hat{m}$ - степень многочлена

T_{crit} - критическое значение Т-статистики для уровня значимости α и стененей свободы $df = n - \hat{m} - 1$

Функция для расчета интервалов указана в приложении №3

Интервалы уровней надежности $\alpha_1 = 0.95$:

[10]:
```
confidence_intervals = model_1.parameters_confidence_intervals(alpha=0.05)

for i in range(len(model_1.params)):
    print(f"theta_{i}: {model_1.params[i]}")
    print(f"Доверительный интервал для коэффициента {i}:␣
 ↪{confidence_intervals[i]}")
    print()
```

```
theta_0: 2.0186212264999988
Доверительный интервал для коэффициента 0: (-1.0376768079343943,
5.074919260934392)
```

интервалы уровней надежности $\alpha_1 = 0.99$

[11]:
```
confidence_intervals = model_1.parameters_confidence_intervals(alpha=0.01)

for i in range(len(model_1.params)):
    print(f"theta_{i}: {model_1.params[i]}")
    print(f"Доверительный интервал для коэффициента {i}:␣
 ↪{confidence_intervals[i]}")
    print()
```

```
theta_0: 2.0186212264999988
Доверительный интервал для коэффициента 0: (-2.0730517484411015,
6.110294201441099)
```

Пункт 3. Доверительные интервалы для полезного сигнала Пусть мы имеем вектор x_{pred}, для которого мы хотим сделать предсказание. Так же есть функция оценки полезного сигнала $\hat{f}$, тогда доверительный интервал для $f(x_{pred})$ при уровне надежности α находится по формуле:

$$\hat{f}(x_{pred}) - T_{crit}\sqrt{\hat{\sigma}^2 * (x_{pred}(X^TX)^{-1}x_{pred}^T)} \leq f(x_{pred}) \leq \hat{f}(x_{pred}) + T_{crit}\sqrt{\hat{\sigma}^2 * (x_{pred}(X^TX)^{-1}x_{pred}^T)}$$

$\hat{\sigma}^2$ - оценка дисперсии ошибок

$$\hat{\sigma}^2 = \frac{(y_i - \hat{y}_i)^2}{n - \hat{m} - 1}$$

где n - кол-во наблюдений, $\hat{m}$ - степень многочлена

T_{crit} - критическое значение T-статистики для уровня значимости α и стененей свободы $df = n - \hat{m} - 1$

Функция для расчета указана в приложении №4

[12]:
```
X_pred = np.linspace(-4, 4, 100)
```

[13]:
```
lower_bound_95, upper_bound_95 = model_1.prediction_interval(X_pred, alpha=0.05)
```

```
95.0% interval: (-1.03767680793439,5.07491926093439)
```

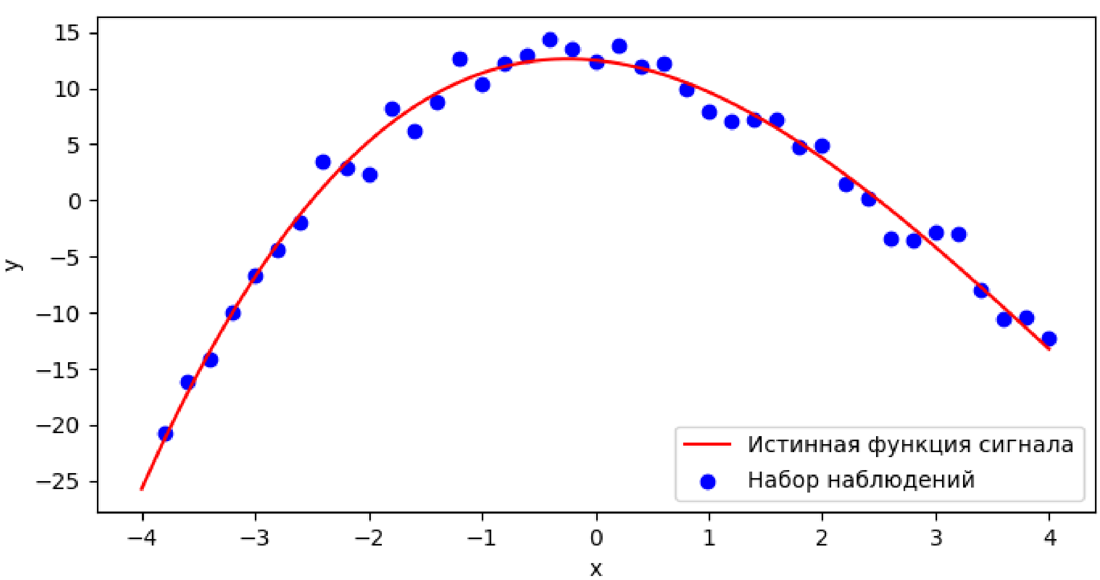
[14]:
```
lower_bound_99, upper_bound_99 = model_1.prediction_interval(X_pred, alpha=0.01)
```

```
99.0% interval: (-2.07305174844110,6.11029420144110)
```

Пункт 4. Графическое представление

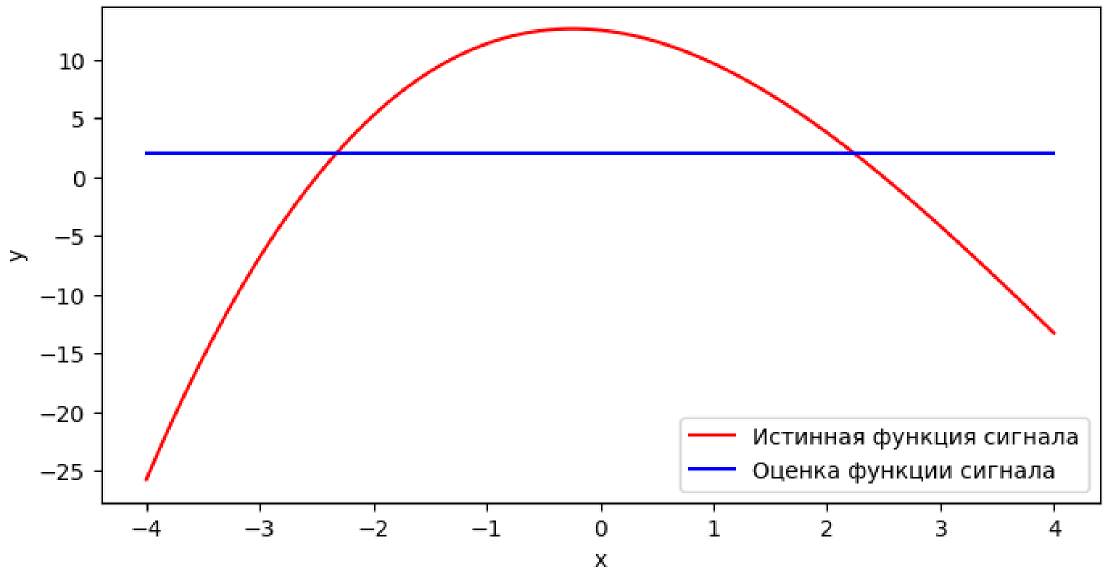
4.1 График истинного полезного сигнала и набора наблюдений

[15]:
```
model_1.vizualize_data_1()
```



4.2 График оценки полезного сигнала, полученного в шаге 1

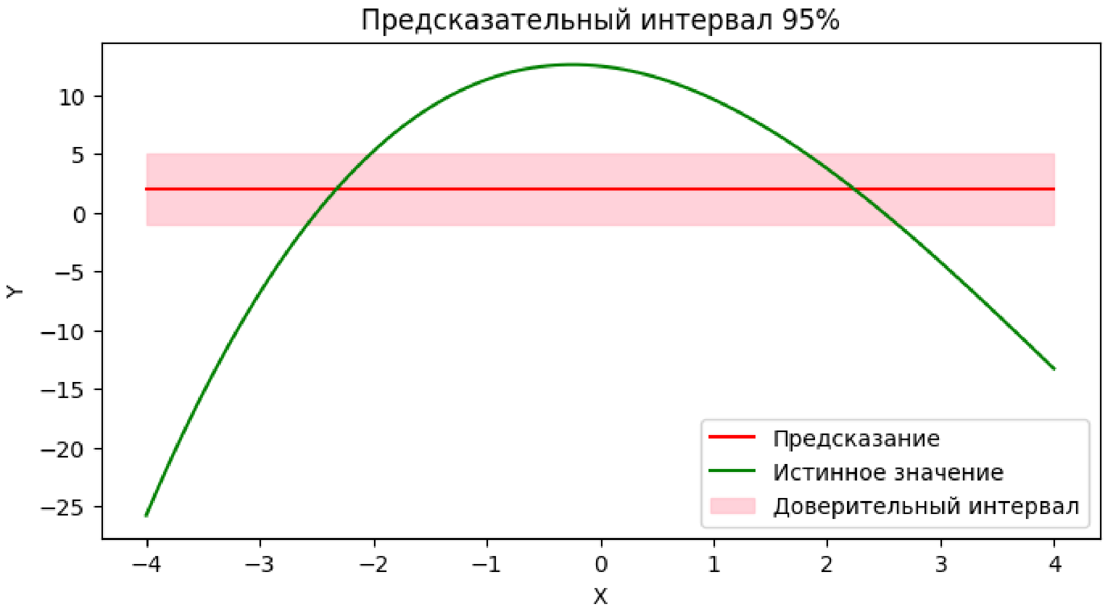
[16]:
```
model_1.vizualize_data_2()
```



4.3 Графики доверительных интервалов полезного сигнала, полученных в шаге 3:

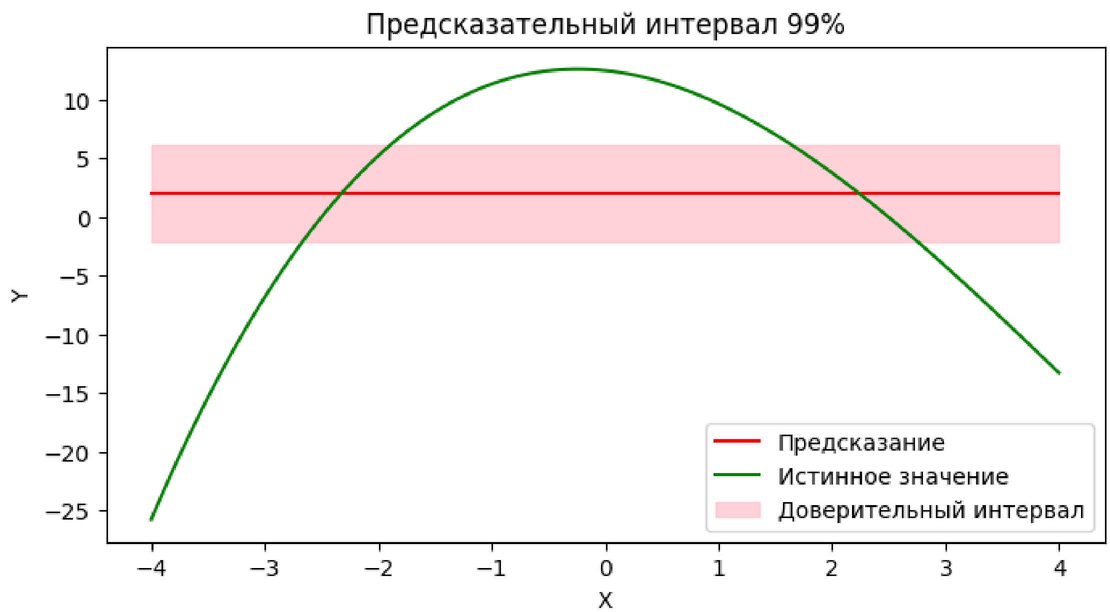
Уровень надежности 0,95:

[17]:
```
# Построение графика
plt.figure(figsize=(8, 4))
plt.plot(X_pred, model_1.predict(X_pred), color='red', label='Предсказание')
plt.plot(X_pred, model_1.phi(X_pred), color='green', label='Истинное значение')
plt.fill_between(X_pred, lower_bound_95, upper_bound_95, color='pink', alpha=0.
 ↪7, label='Доверительный интервал')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Предсказательный интервал 95%')
plt.show()
```



Уровень надежности 0,99:

[18]:
```
# Построение графика
plt.figure(figsize=(8, 4))
plt.plot(X_pred, model_1.predict(X_pred), color='red', label='Предсказание')
plt.plot(X_pred, model_1.phi(X_pred), color='green', label='Истинное значение')
plt.fill_between(X_pred, lower_bound_99, upper_bound_99, color='pink', alpha=0.
↪7, label='Доверительный интервал')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Предсказательный интервал 99%')
plt.show()
```



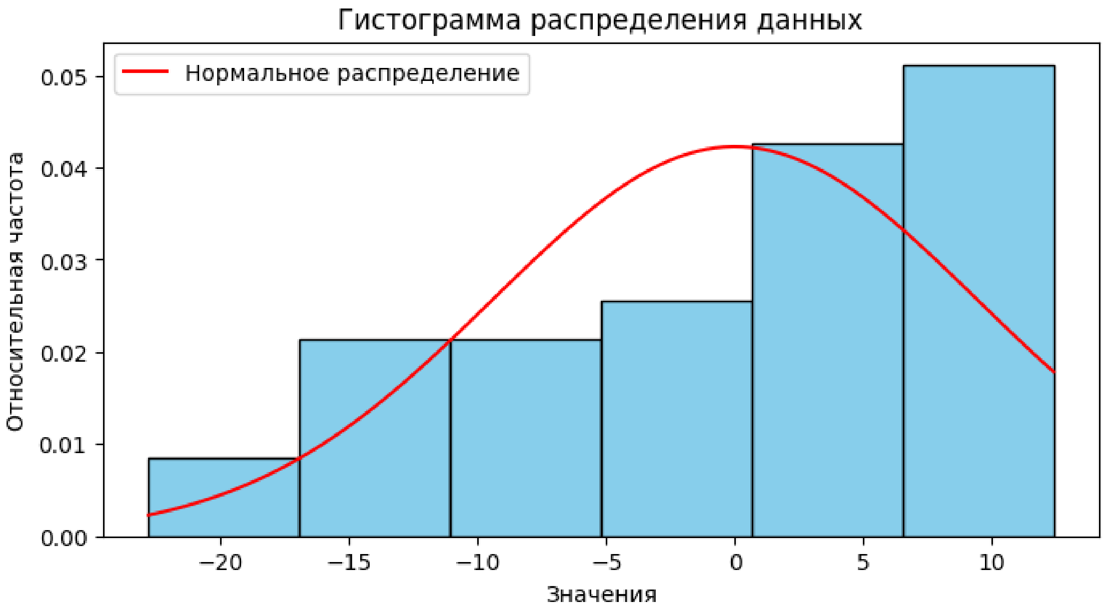
Пункт 5. Гистограмма распределения случайной ошибки Для построения гистограммы расчитаем число столбцов по формуле Стёрджеса:

$$N_{bins} = 1 + [3.322 lgN] = 6$$

Полученная гистограмма:

$$\hat{f}_n(x) = \begin{cases} 0, x \in (-\infty; -22.825) \\ \frac{2}{40 \cdot 5.87}, x \in [-22.825, -16.954) \\ \frac{5}{40 \cdot 5.87}, x \in [-16.954, -11.083) \\ \frac{5}{40 \cdot 5.87}, x \in [-11.083, -5.211) \\ \frac{6}{40 \cdot 5.87}, x \in [-5.211, 0.660) \\ \frac{10}{40 \cdot 5.87}, x \in [0.660, 6.532) \\ \frac{12}{40 \cdot 5.87}, x \in [6.532, 12.403] \\ 0, x \in (12.403; \infty) \end{cases}$$

[19]:
```
p_value, chi2_statistic = model_1.build_hist()
```



Пункт 6. Оценка дисперсии случайной ошибки Для линейной модели регрессии в предположении, что ошибки нормальны МП оценка дисперсии случайной ошибки рассчитывается по формуле:

$$\hat{\sigma^2}_{MLE} = \frac{\sum_{i=1}^{n}(e_i^2)}{n}$$

где $e_i = y_i - \hat{y}_i$ - остатки модели

Доказательство:
Пусть $E \sim N(0, \theta)$, $E = (e_1, e_2, \ldots, e_n)$. Составим функцию правдоподобия:

$$L(E, \theta) = \prod_{i=1}^{n} f_\theta(e_i),\ f_\theta(e_i) = \frac{1}{\sqrt{2\pi\theta}} \exp\left(\frac{-e_i^2}{2\theta}\right) \Rightarrow$$

$$L(E, \theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\theta}} \exp\left(\frac{-e_i^2}{2\theta}\right) = (2\pi\theta)^{-\frac{n}{2}} \exp\left(-\frac{\sum_{i=1}^{n} e_i^2}{2\theta}\right)$$

$$\widetilde{L}(E, \theta) = -\ln(2\pi)^{\frac{n}{2}} - \frac{n}{2}\ln(\theta) - \frac{\sum_{i=1}^{n} e_i^2}{2\theta}$$

$$\frac{\partial \widetilde{L}}{\partial \theta} = -\frac{n}{2\theta} + \frac{\sum_{i=1}^{n} e_i^2}{2\theta^2} = 0 \Rightarrow \theta_{MLE} = \frac{\sum_{i=1}^{n}(e_i^2)}{n} = \hat{\sigma^2}_{MLE}$$

[20]:
```
y_obs = model_1.phi(model_1.x_k)+model_1.eps
y_estimate = model_1.predict(model_1.x_k)
```

```python
# Расчет остатков
residuals = y_obs - y_estimate
print("Вектор остатков:")
print(residuals)
```

```
Вектор остатков:
[-22.82544929 -18.18484184 -16.12068112 -11.98753951  -8.75870514
  -6.45797984  -4.03154248   1.45367593   0.87298478   0.28892138
   6.18166303   4.25885883   6.74906221  10.71114517   8.34891094
  10.23150573  10.87559064  12.40329783  11.55698534  10.37752552
  11.82377419   9.86682749  10.17413001   7.84835174   5.90295949
   4.9872267    5.22028068   5.19134137   2.78335017   2.8494489
  -0.49491763  -1.81273108  -5.42781289  -5.55422134  -4.88814245
  -5.04087559  -9.9463825  -12.58098972 -12.50944301 -14.33556278]
```

[21]:
```python
estimated_variance = np.sum(residuals**2) / (model_1.n)

print(f"Оценка дисперсии случайной ошибки: {estimated_variance}")
```

```
Оценка дисперсии случайной ошибки: 89.04242210895896
```

Пункт 7. Проверка гипотезы о том, что закон распределения ошибки наблюдения является нормальным Основная гипотеза состоит в том, что мы предполагаем, что ошибки распределения имеют нормальное распределение, т.е.

$$H_0 : E \sim N(0, \theta)$$

Альтернативная гипотеза утверждает обратное, т.е.

$$H_A : E \nsim N(0, \theta)$$

Из пункта 6 следует: $\theta = \hat{\sigma^2}_{MLE}$
Нужная нам статистика рассчитывается по формуле:

$$\chi^2 = n \sum_{k=0}^{l} \frac{(\hat{p}_k - p_k)^2}{p_k}$$

где $p_k = F(t_{k+1}) - F(t_k)$

$-\infty = t_o < t_1 < \cdots < t_l < t_{l+1} = \infty$

$F(x)$ в свою очередь является функцией распределения нормального распределения с МП оценками параметров (т.е. выборочное среднее и выборочная дисперсия)

Алгоритм проверки гипотезы 1. Расчитать статистику 2. Расчитать `p-value` для полученной статистики для χ^2 распределения с $df = \hat{n} - 1 = 6$ степеней свободы (так как кол-во ненулевых промежутков = 8, $\hat{n} = 8 - 1 = 7$, и -1, т.к. два параметра у нормального распредления) 3. Если `p-value` меньше α (в нашем случае $\alpha = 0.05$), то отклоняем нулевую гипотезу. Иначе признаем нулевую гипотезу верной.

[22]:
```python
# Уровень значимости
alpha = 0.05

# Вывод результатов
print(f"Статистика хи-квадрат: {chi2_statistic}")
print(f"P-значение: {p_value}")

# Проверяем гипотезу
if p_value < alpha:
    print("Отклоняем гипотезу о нормальности распределения остатков.")
else:
    print("Не отклоняем гипотезу о нормальности распределения остатков.")
```

```
Статистика хи-квадрат: 13.398849090520075
P-значение: 0.037121741074052306
Отклоняем гипотезу о нормальности распределения остатков.
```

Случай 2

$m = 2, \varepsilon_k \sim R(-3\sigma, 3\sigma)$

Инициализируем модель с заданными параметрами для первого случая

[23]:
```python
model_2 = Model(m=2, theta=[12.5, -1.0, -2.0, 0.16], sigma=1.6, normal_eps=False)
```

[24]:
```python
print("Вектор ошибок:\n")
print(model_2.eps)
```

```
Вектор ошибок:

[ 0.35186123  2.50308426  3.30793536  0.92420499  3.34212749  2.51403782
 -1.56516637 -0.1159193   3.61458643  0.48827141  0.69511566 -0.72858629
  3.11903726 -1.05148148  0.21004824 -1.71037372 -3.77297942  0.62989951
 -3.07461475  2.86154738 -1.59508467 -2.18020186 -0.29266487 -1.4560514
 -3.70726325 -2.67668282  0.41980027  2.94194947  0.53142816 -3.40646812
  0.85147408 -1.43709791 -0.93867701  1.94193616 -1.63086416 -1.89904388
 -2.00411181  0.0882728  -0.59633548  1.61174472]
```

[25]:
```python
print("список X, на которых строятся наблюдения:\n")
print(model_2.x_k)
```

```
список X, на которых строятся наблюдения:

[-3.8 -3.6 -3.4 -3.2 -3.  -2.8 -2.6 -2.4 -2.2 -2.  -1.8 -1.6 -1.4 -1.2
 -1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8  1.   1.2  1.4  1.6
  1.8  2.   2.2  2.4  2.6  2.8  3.   3.2  3.4  3.6  3.8  4. ]
```

[26]:
```python
print("Полученные наблюдения:\n")
print(model_2.y_obs)
```

```
Полученные наблюдения:

[-1.22281388e+01 -7.31691574e+00 -3.91206464e+00 -3.85579501e+00
  8.42127490e-01  2.13403782e+00  1.48336300e-02  3.26408070e+00
  8.63458643e+00  6.98827141e+00  8.51511566e+00  8.25141371e+00
  1.30990373e+01  9.76851852e+00  1.17100482e+01  1.03096263e+01
  8.60702058e+00  1.32098995e+01  9.54538525e+00  1.53615474e+01
  1.06249153e+01  9.59979814e+00  1.08873351e+01  8.96394860e+00
  5.79273675e+00  5.74331718e+00  7.59980027e+00  8.72194947e+00
  4.75142816e+00 -9.06468120e-01  1.47147408e+00 -2.85709791e+00
 -4.55867701e+00 -4.03806384e+00 -1.01308642e+01 -1.30790439e+01
 -1.60241118e+01 -1.69317272e+01 -2.07763355e+01 -2.18882553e+01]
```

Пункт 1. Вычисление порядка многочлена для модели, нахождение оценки параметров Проверка значимости коэффициентов регрессии означает проверку основной гипотезы об их значимом отличии от нуля.

Основная гипотеза состоит в предположении о незначимости коэффициента модели регрессии, т. е.

$$H_0 : \hat{\theta}_{\hat{m}} = 0$$

Обратная или конкурирующая гипотеза состоит в предположении о значимости коэффициентов модели регрессии, т. е.

$$H_1 : \hat{\theta}_{\hat{m}} \neq 0$$

Для проверки гипотезы на уровне значимости α используем статистический критерий $T_i = \frac{\hat{\theta}_i}{\mu_i}$, где $\hat{\theta}_i$ – оценка параметра модели, а μ_i – ее стандартная ошибка

$$\mu_i = \sqrt{\hat{\sigma}^2 * (X^T X)^{-1}_{ii}}$$

$\hat{\sigma}^2$ - оценка дисперсии ошибок

$$\hat{\sigma}^2 = \frac{(y_i - \hat{y}_i)^2}{n - \hat{m} - 1}$$

где n - кол-во наблюдений, $\hat{m}$ - степень многочлена

Для нахождения степени многочлена будем итеративно перебирать степень $\hat{m}$ пока не найдем такую, что при $\hat{m} + 1$ выполняется H_0.

Функция для итеративного T-теста описана в приложении №2

[27]:
```python
from model import iterative_t_test
```

[28]:
```python
m_hat = 1
print('================================================================================')
print('m_estimate = ', m_hat)
while iterative_t_test(model_2.x_k, model_2.phi(model_2.x_k)+model_2.eps, m_hat):
    m_hat+=1
    
 ↪print('================================================================================')
```

```
    print('m_estimate = ', m_hat)

m_hat -= 1
```

```
================================================================================
m_estimate =  1
theta_1 является статистически значимым.
================================================================================
m_estimate =  2
theta_2 является статистически значимым.
================================================================================
m_estimate =  3
theta_3 не является статистически значимым.
```

Так как при $\hat{m} = 3$: $\hat{\theta}_3$ не является статистически значимым $\Rightarrow$ полагаем, что $\hat{m} = 2$

Расчитаем коэфициенты:

[29]:
```
model_2.fit(m_hat)
for i in range(len(model_2.params)):
    print(f"theta_{i} = {model_2.params[i]}")
```

```
theta_0 = 11.708747580268767
theta_1 = -1.3433713936254033
theta_2 = -1.858931289823263
```

Воспользуемся специализированной статистической библиотекой на `python` и проверим результы полученные в библиотеке с теми, которые получили на прошлом этапе

[30]:
```
X = np.vander(model_2.x_k, m_hat+1, increasing=True)
model = sm.OLS(model_2.y_obs, sm.add_constant(X))
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.966
Model:                            OLS   Adj. R-squared:                  0.965
Method:                 Least Squares   F-statistic:                     531.8
Date:                Mon, 04 Dec 2023   Prob (F-statistic):           5.52e-28
Time:                        15:27:39   Log-Likelihood:                -80.470
No. Observations:                  40   AIC:                             166.9
Df Residuals:                      37   BIC:                             172.0
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         11.7087      0.446     26.249      0.000      10.805      12.613
x1            -1.3434      0.129    -10.380      0.000      -1.606      -1.081
```

```
x2            -1.8589      0.062    -29.770      0.000      -1.985      -1.732
==============================================================================
Omnibus:                        1.546   Durbin-Watson:                   2.254
Prob(Omnibus):                  0.462   Jarque-Bera (JB):                1.410
Skew:                           0.332   Prob(JB):                        0.494
Kurtosis:                       2.363   Cond. No.                         10.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

Пункт 2. Доверительные интервалы для параметров Для рассчета доверительных интервалов для параметров воспользуемся формулой:

$$\theta_i \in (\hat{\theta}_i - T_{crit} * \mu_i; \hat{\theta}_i + T_{crit} * \mu_i)$$

где μ_i - это стандартная ошибка параметра

$$\mu_i = \sqrt{\hat{\sigma}^2 * (X^T X)^{-1}_{ii}}$$

$\hat{\sigma}^2$ - оценка дисперсии ошибок

$$\hat{\sigma}^2 = \frac{(y_i - \hat{y}_i)^2}{n - \hat{m} - 1}$$

где n - кол-во наблюдений, $\hat{m}$ - степень многочлена

T_{crit} - критическое значение T-статистики для уровня значимости α и стеneней свободы $df = n - \hat{m} - 1$

Функция для расчета интервалов указана в приложении №3

Интервалы уровней надежности $\alpha_1 = 0.95$:

[31]:
```
confidence_intervals = model_2.parameters_confidence_intervals(alpha=0.05)

for i in range(len(model_2.params)):
    print(f"theta_{i}: {model_2.params[i]}")
    print(f"Доверительный интервал для коэффициента {i}:␣
 ↪{confidence_intervals[i]}")
    print()
```

```
theta_0: 11.708747580268767
Доверительный интервал для коэффициента 0: (10.804934701472053,
12.612560459065481)

theta_1: -1.3433713936254033
Доверительный интервал для коэффициента 1: (-1.6056121933414391,
-1.0811305939093674)
```

```
theta_2: -1.858931289823263
Доверительный интервал для коэффициента 2: (-1.9854538945815796,
-1.7324086850649465)
```

интервалы уровней надежности $\alpha_1 = 0.99$

[32]:
```
confidence_intervals = model_2.parameters_confidence_intervals(alpha=0.01)

for i in range(len(model_2.params)):
    print(f"theta_{i}: {model_2.params[i]}")
    print(f"Доверительный интервал для коэффициента {i}:␣
 ↪{confidence_intervals[i]}")
    print()
```

```
theta_0: 11.708747580268767
Доверительный интервал для коэффициента 0: (10.497499676372819,
12.919995484164716)

theta_1: -1.3433713936254033
Доверительный интервал для коэффициента 1: (-1.6948142933934747,
-0.9919284938573318)

theta_2: -1.858931289823263
Доверительный интервал для коэффициента 2: (-2.0284909889647436,
-1.6893715906817826)
```

Пункт 3. Доверительные интервалы для полезного сигнала Пусть мы имеем вектор x_{pred}, для которого мы хотим сделать предсказание. Так же есть функция оценки полезного сигнала $\hat{f}$, тогда доверительный интервал для $f(x_{pred})$ при уровне надежности α находится по формуле:

$$\hat{f}(x_{pred}) - T_{crit}\sqrt{\hat{\sigma}^2 * (x_{pred}(X^TX)^{-1}x_{pred}^T)} \leq f(x_{pred}) \leq \hat{f}(x_{pred}) + T_{crit}\sqrt{\hat{\sigma}^2 * (x_{pred}(X^TX)^{-1}x_{pred}^T)}$$

$\hat{\sigma}^2$ - оценка дисперсии ошибок

$$\hat{\sigma}^2 = \frac{(y_i - \hat{y_i})^2}{n - \hat{m} - 1}$$

где n - кол-во наблюдений, $\hat{m}$ - степень многочлена

T_{crit} - критическое значение T-статистики для уровня значимости α и стененей свободы $df = n - \hat{m} - 1$

Функция для расчета указана в приложении №4

[33]:
```
X_pred = np.linspace(-4, 4, 100)
```

[34]:
```
lower_bound_95, upper_bound_95 = model_2.prediction_interval(X_pred, alpha=0.05)
```

```
95.0% interval: (-1.85893128982326*x**2 - 1.3433713936254*x -
3.8112069705003*sqrt(x**2*(0.00110207507511744*x**2 - 0.000220415015023488*x -
0.00586303939962477) - 0.00586303939962477*x**2 + x*(-0.000220415015023488*x**2
+ 0.00473451452270451*x + 0.000703564727954975) + 0.000703564727954975*x +
0.0562382739212008) + 11.7087475802688,-1.85893128982326*x**2 -
1.3433713936254*x + 3.8112069705003*sqrt(x**2*(0.00110207507511744*x**2 -
0.000220415015023488*x - 0.00586303939962477) - 0.00586303939962477*x**2 +
x*(-0.000220415015023488*x**2 + 0.00473451452270451*x + 0.000703564727954975) +
0.000703564727954975*x + 0.0562382739212008) + 11.7087475802688)
```

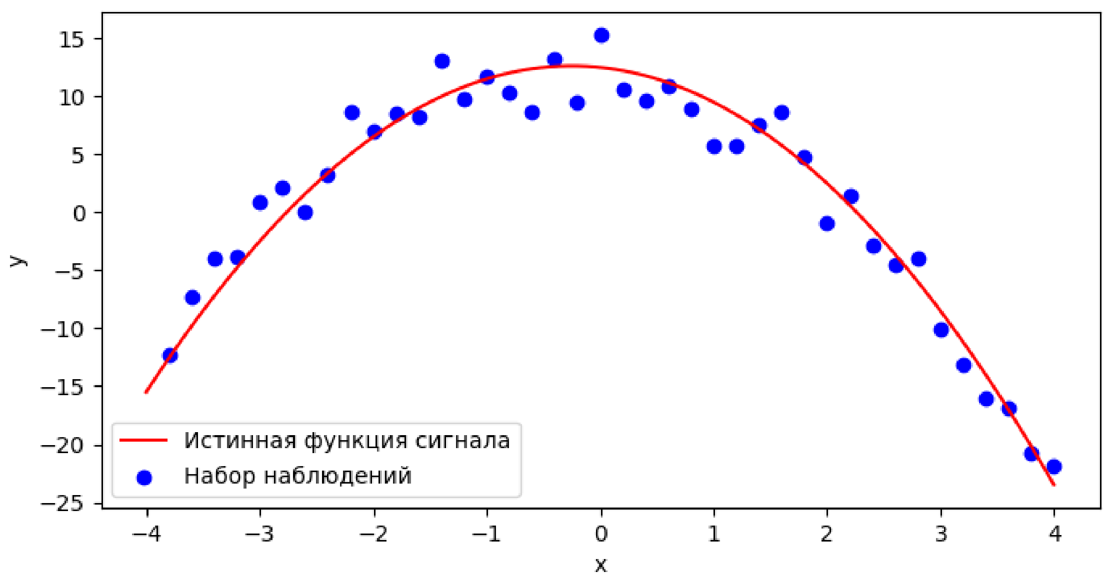
[35]:
```
lower_bound_99, upper_bound_99 = model_2.prediction_interval(X_pred, alpha=0.01)
```

```
99.0% interval: (-1.85893128982326*x**2 - 1.3433713936254*x -
5.10760198557695*sqrt(x**2*(0.00110207507511744*x**2 - 0.000220415015023488*x -
0.00586303939962477) - 0.00586303939962477*x**2 + x*(-0.000220415015023488*x**2
+ 0.00473451452270451*x + 0.000703564727954975) + 0.000703564727954975*x +
0.0562382739212008) + 11.7087475802688,-1.85893128982326*x**2 -
1.3433713936254*x + 5.10760198557695*sqrt(x**2*(0.00110207507511744*x**2 -
0.000220415015023488*x - 0.00586303939962477) - 0.00586303939962477*x**2 +
x*(-0.000220415015023488*x**2 + 0.00473451452270451*x + 0.000703564727954975) +
0.000703564727954975*x + 0.0562382739212008) + 11.7087475802688)
```

Пункт 4. Графическое представление

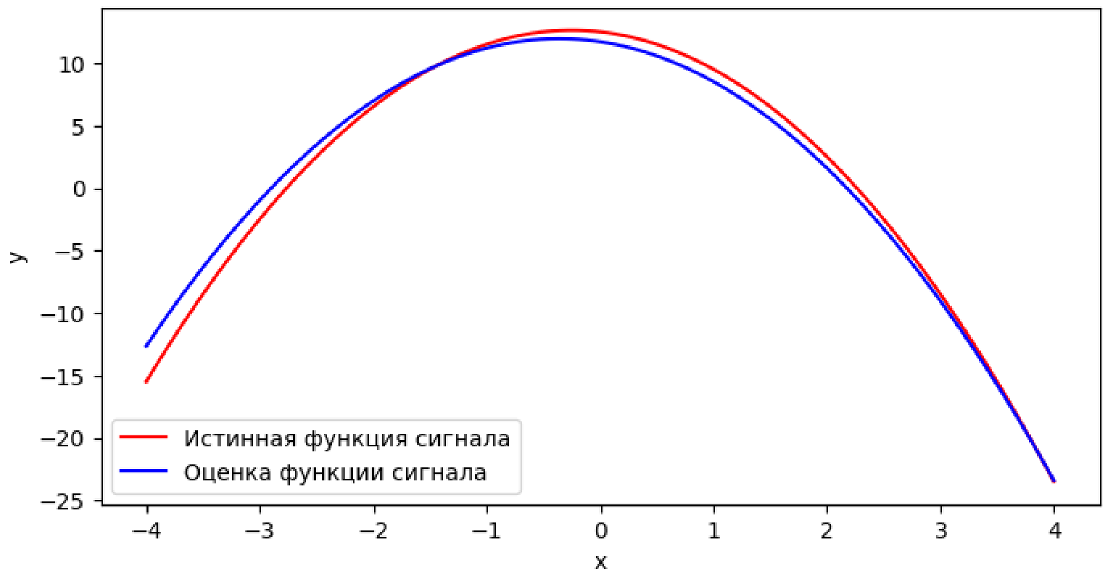
4.1 График истинного полезного сигнала и набора наблюдений

[36]:
```
model_2.vizualize_data_1()
```



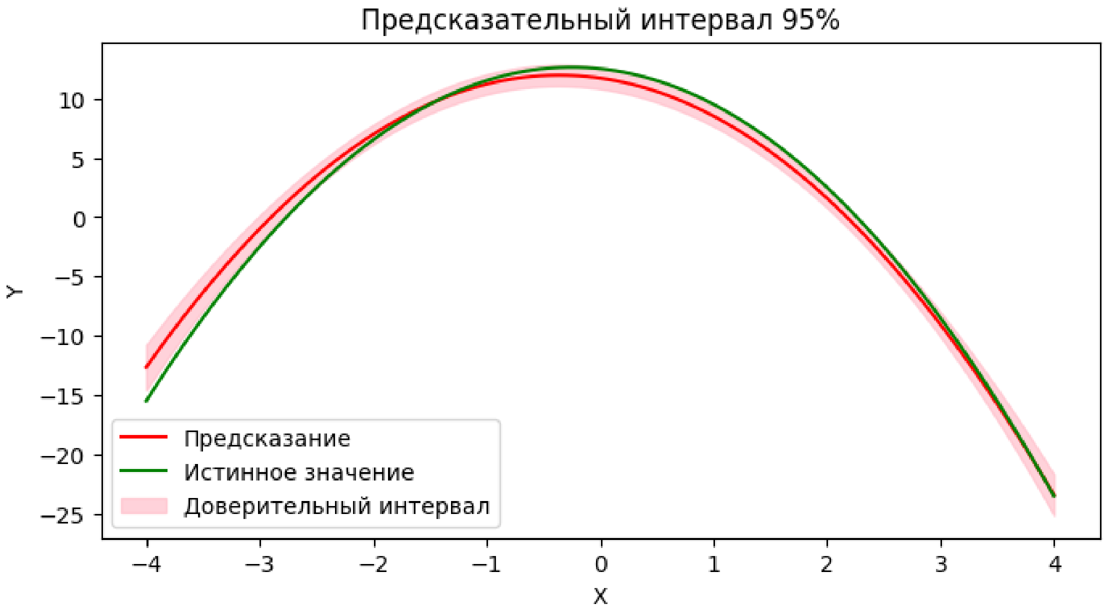
4.2 График оценки полезного сигнала, полученного в шаге 1

[37]:
```
model_2.vizualize_data_2()
```



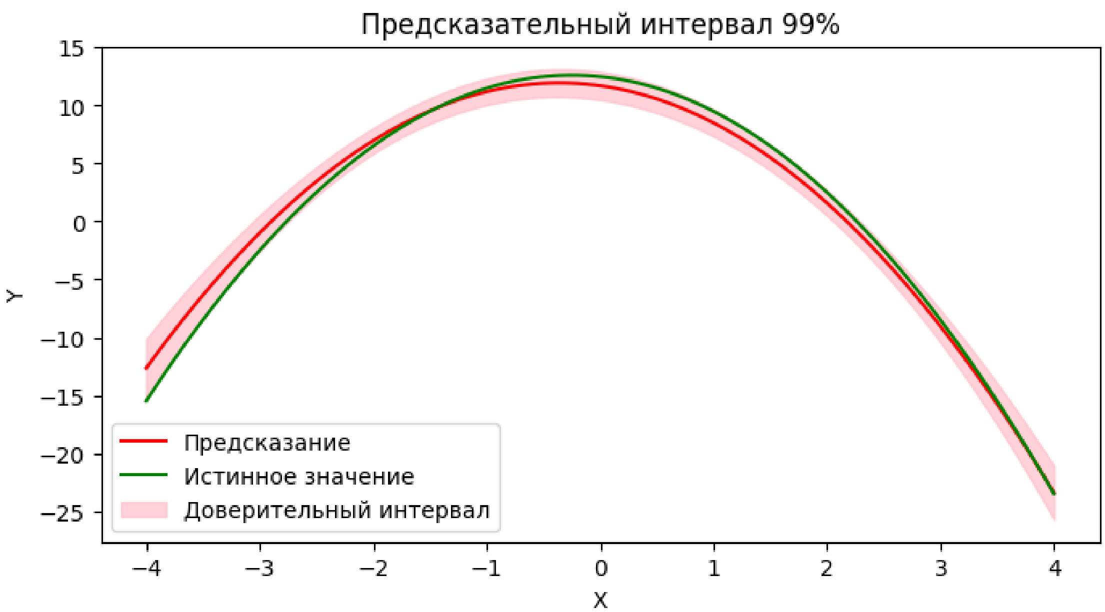
4.3 Графики доверительных интервалов полезного сигнала, полученных в шаге 3:
Уровень надежности 0,95:

[38]:
```
# Построение графика
plt.figure(figsize=(8, 4))
plt.plot(X_pred, model_2.predict(X_pred), color='red', label='Предсказание')
plt.plot(X_pred, model_2.phi(X_pred), color='green', label='Истинное значение')
plt.fill_between(X_pred, lower_bound_95, upper_bound_95, color='pink', alpha=0.
↪7, label='Доверительный интервал')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Предсказательный интервал 95%')
plt.show()
```



Уровень надежности 0,99:

[39]:
```
# Построение графика
plt.figure(figsize=(8, 4))
plt.plot(X_pred, model_2.predict(X_pred), color='red', label='Предсказание')
plt.plot(X_pred, model_2.phi(X_pred), color='green', label='Истинное значение')
plt.fill_between(X_pred, lower_bound_99, upper_bound_99, color='pink', alpha=0.
↪7, label='Доверительный интервал')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Предсказательный интервал 99%')
plt.show()
```



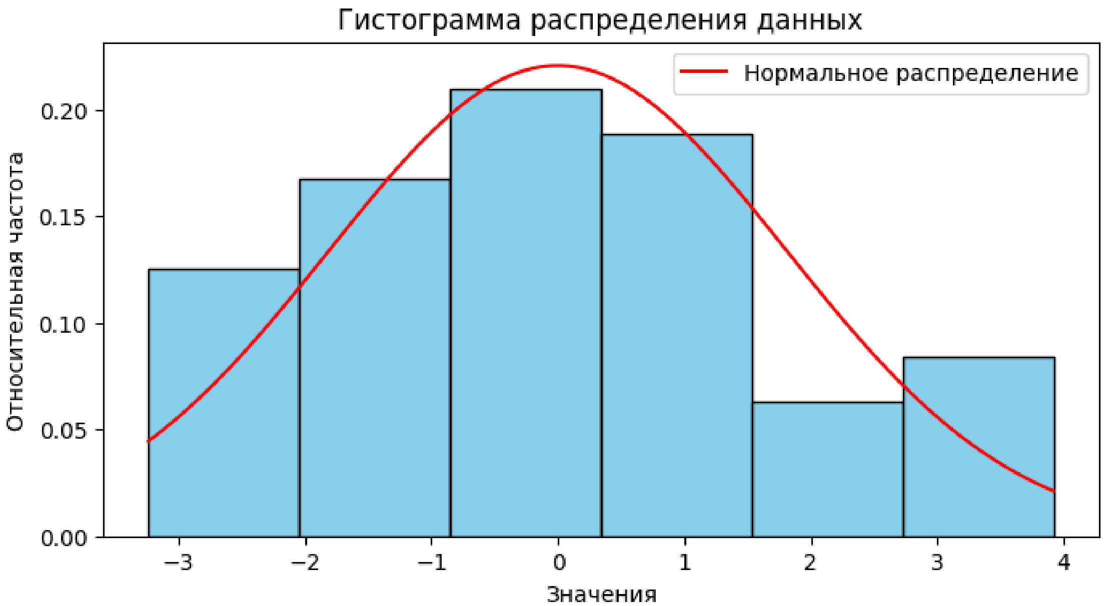
Пункт 5. Гистограмма распределения случайной ошибки Для построения гистограммы расчитаем число столбцов по формуле Стёрджеса:

$$N_{bins} = 1 + [3.322 lgN] = 6$$

Полученная гистограмма:

$$\hat{f}_n(x) = \begin{cases} 0, x \in (-\infty; -3.239) \\ \frac{6}{40 \cdot 3.921}, x \in [-3.239, -2.045) \\ \frac{8}{40 \cdot 3.921}, x \in [-2.045, -0.852) \\ \frac{10}{40 \cdot 3.921}, x \in [-0.852, 0.341) \\ \frac{9}{40 \cdot 3.921}, x \in [0.341, 1.535) \\ \frac{3}{40 \cdot 3.921}, x \in [1.535, 2.728) \\ \frac{4}{40 \cdot 3.921}, x \in [2.728, 3.921] \\ 0, x \in (3.921; \infty) \end{cases}$$

[40]:
```
p_value, chi2_statistic = model_2.build_hist()
```



Пункт 6. Оценка дисперсии случайной ошибки Для линейной модели регрессии в предположении, что ошибки нормальны МП оценка дисперсии случайной ошибки рассчитывается по формуле:

$$\hat{\sigma^2}_{MLE} = \frac{\sum_{i=1}^{n}(e_i^2)}{n}$$

где $e_i = y_i - \hat{y}_i$ - остатки модели

Доказательство:
Пусть $E \sim N(0, \theta)$, $E = (e_1, e_2, \dots, e_n)$. Составим функцию правдоподобия:

$$L(E, \theta) = \prod_{i=1}^{n} f_\theta(e_i),\ f_\theta(e_i) = \frac{1}{\sqrt{2\pi\theta}} \exp\left(\frac{-e_i^2}{2\theta}\right) \Rightarrow$$

$$L(E, \theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\theta}} \exp\left(\frac{-e_i^2}{2\theta}\right) = (2\pi\theta)^{-\frac{n}{2}} \exp\left(-\frac{\sum_{i=1}^{n} e_i^2}{2\theta}\right)$$

$$\widetilde{L}(E, \theta) = -\ln(2\pi)^{\frac{n}{2}} - \frac{n}{2}\ln(\theta) - \frac{\sum_{i=1}^{n} e_i^2}{2\theta}$$

$$\frac{\partial \widetilde{L}}{\partial \theta} = -\frac{n}{2\theta} + \frac{\sum_{i=1}^{n} e_i^2}{2\theta^2} = 0 \Rightarrow \theta_{MLE} = \frac{\sum_{i=1}^{n}(e_i^2)}{n} = \hat{\sigma^2}_{MLE}$$

[41]:
```
y_obs = model_2.phi(model_1.x_k)+model_2.eps
y_estimate = model_2.predict(model_2.x_k)
```

```python
# Расчет остатков
residuals = y_obs - y_estimate
print("Вектор остатков:")
print(residuals)
```

```
Вектор остатков:
[-2.19872982  0.22994918  1.30097075 -0.82787464  1.83364734  1.23787165
 -2.62030405 -0.961314    2.96764923  0.0285062   0.41123695 -0.847864
  3.15307506 -0.87541368  0.51686056 -1.28410239 -3.23853457  1.26123238
 -2.35767936  3.6527998  -0.74080072 -1.27417188  0.65382565 -0.48038584
 -2.71370815 -1.67652367  1.41527797  3.92146022  1.48368647 -2.49274775
  1.71537101 -0.63430992 -0.20828345  2.58864979 -1.07911595 -1.45354659
 -1.67615094  0.28741175 -0.53730394  1.51938335]
```

[42]:
```python
estimated_variance = np.sum(residuals**2) / (model_2.n)

print(f"Оценка дисперсии случайной ошибки: {estimated_variance}")
```

```
Оценка дисперсии случайной ошибки: 3.2726940723286426
```

Пункт 7. Проверка гипотезы о том, что закон распределения ошибки наблюдения является нормальным Для проверки гипотезы о том, что закон распределения ошибки наблюдения является нормальным воспользуемся χ^2-критерием Пирсона

Основная гипотеза состоит в том, что мы предполагаем, что ошибки распределения имеют нормальное распределение, т.е.

$$H_0 : E \sim N(0, \theta)$$

Альтернативная гипотеза утверждает обратное, т.е.

$$H_A : E \nsim N(0, \theta)$$

Из пункта 6 следует: $\theta = \hat{\sigma^2}_{MLE}$
Нужная нам статистика рассчитывается по формуле:

$$\chi^2 = n \sum_{k=0}^{l} \frac{(\hat{p}_k - p_k)^2}{p_k}$$

где $p_k = F(t_{k+1}) - F(t_k)$

$-\infty = t_o < t_1 < \cdots < t_l < t_{l+1} = \infty$

$F(x)$ в свою очередь является функцией распределения нормального распределения с МП оценками параметров (т.е. выборочное среднее и выборочная дисперсия)

Алгоритм проверки гипотезы 1. Расчитать статистику 2. Расчитать `p-value` для полученной статистики для χ^2 распределения с $df = \hat{n} - 1 = 6$ степеней свободы (так как кол-во ненулевых промежутков $= 8$, $\hat{n} = 8 - 1 = 7$, и -1, т.к. один параметр у нормального распредления) 3. Если `p-value` меньше α (в нашем случае $\alpha = 0.05$), то отклоняем нулевую гипотезу. Иначе признаем нулевую гипотезу верной.

[43]:
```python
# Уровень значимости
alpha = 0.05

# Вывод результатов
print(f"Статистика хи-квадрат: {chi2_statistic}")
print(f"P-значение: {p_value}")

# Проверяем гипотезу
if p_value < alpha:
    print("Отклоняем гипотезу о нормальности распределения остатков.")
else:
    print("Не отклоняем гипотезу о нормальности распределения остатков.")
```

```
Статистика хи-квадрат: 6.449577106502813
P-значение: 0.3747543818200002
Не отклоняем гипотезу о нормальности распределения остатков.
```

Приложения

Приложение №1

```python
class Model:
    '''
    ## Класс для модели наблюдений.
    ### Описывается следющим
    и параметрами:
    - m: теоретический порядок многочлена
    - thetha: теоретические параметры модели
    - sigma: дисперсия отклонений ошибок наблюдений
    - normal_eps: булевая переменная. Принимает True если ошибки распределены␣
↪нормально, иначе False (по умолчанию True)
    - n: количество наблюдений (по умолчанию 40)
    '''
    def __init__(self, m:int, theta:list[float], sigma:float, normal_eps:
↪bool=True, n:int=40):
        self.m = m
        self.n = n
        self.theta = theta
        np.random.seed(42)
        self.sigma = sigma
        if normal_eps:
            self.eps = np.random.normal(0, sigma, n)
        else:
            self.eps = np.random.uniform(-3*np.sqrt(sigma), 3*np.sqrt(sigma), n)
        self.x_k = np.array([-4 + k*(8/n) for k in range(1,n+1)])

        self.estimate_coefficients = []


    @property
    def params(self):
        assert len(self.estimate_coefficients) != 0, "Коэффициенты не найдены"
        return self.estimate_coefficients

    @property
    def mse(self) -> float:
        y_obs = self.phi(self.x_k)+self.eps
        y_pred = self.predict(self.x_k)
        residuals = y_obs - y_pred
        mse = np.sum(residuals ** 2) / (self.n - len(self.params))
        return mse

    @property
    def y_obs(self):
        return self.phi(self.x_k)+self.eps
```

```python
    def phi(self, x) -> float:
        '''
        ## Теоретическая функция полезного сигнала
        '''
        return sum([self.theta[i]*(x**i) for i in range(self.m+1)])

    def predict(self, x) -> float:
        '''
        ## Функция оценки полезного сигнала
        '''
        return sum([self.params[i]*(x**i) for i in range(len(self.estimate_
↪coefficients))])

    def fit(self, degree:int):
        '''
        ## Функция для оценки коэффициентов многочлена
        ### Параметры:
        - degree: порядок многочлена
        '''
        X = np.vander(self.x_k, degree+1, increasing=True)
        y = self.phi(self.x_k)+self.eps
        self.estimate_coefficients = np.linalg.inv(X.T @ X) @ X.T @ y
        return self.estimate_coefficients


    def parameters_confidence_intervals(self, alpha = 0.05):
        '''
        ## Расчет доверительного интервала для найденных параметров
        ### Параметры:
        - alpha: уровень значимости (по умолчанию 0.05)
        '''

        # расчет стандартных ошибок параметров
        X = np.vander(self.x_k, len(self.params), increasing=True)
        std_errors = np.sqrt(np.diagonal(self.mse * np.linalg.inv(np.dot(X.T, X))))

        # Определение степеней свободы
        degrees_of_freedom = self.n - len(self.params)

        # Определение критического значения t для уровня значимости и степеней␣
↪свободы
        t_critical = t.ppf(1 - alpha/2, df=degrees_of_freedom)

        # Построение доверительных интервалов
        confidence_intervals = []
        for i in range(len(self.params)):
```

```
            lower_bound = self.params[i] - t_critical * std_errors[i]
            upper_bound = self.params[i] + t_critical * std_errors[i]
            confidence_intervals.append((lower_bound, upper_bound))

        return confidence_intervals

    def prediction_interval(self, x_pred, alpha=0.05):
        '''
        ## Расчет предсказательного интервала
        ### Параметры:
        - x_pred: значение для предсказания
        - alpha: уровень значимости (по умолчанию 0.05)
        '''

        def vandermonde_matrix(variables, n):
            """
            Создает вендермондову матрицу для заданных переменных и степени n.
            """
            return Matrix([[var**i for i in range(n)] for var in variables])

        # Определяем матрицу
        x = symbols('x')
        A = vandermonde_matrix([x], len(self.params))

        X = np.vander(self.x_k, len(self.params), increasing=True)

        # Критическое значение t для выбранного уровня доверия
        t_value = t.ppf(1 - alpha / 2, df=self.n-len(self.params))

        # Стандартная ошибка предсказания для нового наблюдения
        B = np.linalg.inv(np.dot(X.T, X))

        f = ((self.mse * ((A * B) * A.T)).applyfunc(sqrt))

        betas = Matrix(self.params)
        pred = A * betas
        low_edge = pred - t_value * f
        high_edge = pred + t_value * f
        print(f'{(1-alpha)*100}% interval: ({low_edge[0]},{high_edge[0]})' )
        l, h = [], []
        for x_val in x_pred:
            l.append(low_edge.subs({x:x_val})[0])
            h.append(high_edge.subs({x:x_val})[0])

        return np.array(l).astype(float), np.array(h).astype(float)

    def vizualize_data_1(self):
        '''
```

```
        ## Функция визуализации истинной функции и набора наблюдений
        '''
        x = np.linspace(-4, 4, 100)
        plt.figure(figsize=(8, 4))
        plt.plot(x, self.phi(x), color='red', label='Истинная функция сигнала')
        plt.scatter(self.x_k, self.phi(self.x_k)+self.eps, color='blue', label=
↪'Набор наблюдений')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
        plt.show()


    def vizualize_data_2(self):
        '''
        ## Функция визуализации истинной функции и оценки функции
        '''
        x = np.linspace(-4, 4, 100)
        plt.figure(figsize=(8, 4))
        plt.plot(x, self.phi(x), color='red', label='Истинная функция сигнала')
        plt.plot(x, self.predict(x), color='blue', label='Оценка функции сигнала')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
        plt.show()


    def build_hist(self):
        '''
        ## Построение гистограммы распределения остатков
        '''

        y_estimate = self.predict(self.x_k)

        # Расчет остатков
        residuals = self.y_obs - y_estimate

        # Выбор количества интервалов (бинов)
        num_bins = int(3.32*np.log10(self.n)+1)

        # Разделение данных на интервалы
        hist, bin_edges = np.histogram(residuals, bins=num_bins, density=True)
        hist_for_print, _ = np.histogram(residuals, bins=num_bins, density=False)
        # Вычисление ширины интервала
        bin_width = bin_edges[1] - bin_edges[0]
        expected_freq = np.diff([0]+ list(norm.cdf(bin_edges, loc=0, scale=(np.
↪sum(residuals**2) / (self.n))))+[1])
```

```
        estimated_freq = [0] + list(hist*bin_width) + [0]

        chi2_statistic = len(self.x_k)*np.sum((estimated_freq - expected_freq) **␣
↪2 / expected_freq)
        p_value = 1 - np.sum(chi2.cdf(chi2_statistic, df=len(bin_edges) - 1))

        plt.figure(figsize=(8, 4))
        plt.hist(residuals, bins=num_bins, density=True, color='skyblue',␣
↪edgecolor='black')
        x_range = np.linspace(bin_edges[0], bin_edges[-1], 100)
        normal_distribution = norm.pdf(x_range, 0, np.std(residuals))
        plt.plot(x_range, normal_distribution, color='red', label='Нормальное␣
↪распределение')

        # Добавление подписей и оформление
        plt.title('Гистограмма распределения данных')
        plt.xlabel('Значения')
        plt.ylabel('Относительная частота')
        plt.legend()
        plt.show()

        return p_value, chi2_statistic
```

Приложение №2

```
def iterative_t_test(X, y, n, alpha=0.05):

    X_with_intercept = np.vander(X, n+1, increasing=True)

    num_samples, num_features = X_with_intercept.shape

    # Инициализация массива для хранения t-статистик
    t_stats = np.zeros(num_features)

    # Оценка коэффициентов регрессии с помощью метода наименьших квадратов
    betas = np.linalg.inv(X_with_intercept.T.dot(X_with_intercept)).dot(X_with_
↪intercept.T).dot(y)

    # Оценка дисперсии ошибок
    y_pred = X_with_intercept.dot(betas)
    residuals = y - y_pred
    error_variance = np.sum(residuals ** 2) / (num_samples - num_features)

    # Вычисление стандартных ошибок оценок коэффициентов
    se_betas = np.sqrt(np.diag(error_variance * np.linalg.inv(X_with_intercept.T.
↪dot(X_with_intercept))))

    # Вычисление t-статистик
```

```python
t_stats = betas / se_betas

# Вычисление p-значений
degrees_freedom = num_samples - num_features
p_values = 2 * (1 - t.cdf(np.abs(t_stats), df=degrees_freedom))

flag = True
# Проверка на значимость коэффициента при заданном уровне значимости
if p_values[n] < alpha:
    print(f"theta_{n} является статистически значимым.")
else:
    print(f"theta_{n} не является статистически значимым.")
    flag = False
return flag
```