Математическая статистика

Практическое задание 5

В данном задании предлагается провести некоторое исследование модели линейной регрессии и критериев для проверки статистических гипотез, в частности применить этим модели к реальным данным.

Правила:

- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма "`[номер группы] Фамилия Имя - Задание 5`". Квадратные скобки обязательны. Вместо `Фамилия Имя` нужно подставить свои фамилию и имя.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `5.N.ipynb` и `5.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- Никакой код из данного задания при проверке запускаться не будет.
- Некоторые задачи отмечены символом *. Эти задачи являются дополнительными. Успешное выполнение большей части таких задач (за все задания) является необходимым условием получения бонусного балла за практическую часть курса.
- Баллы за каждую задачу указаны далее. Если сумма баллов за задание меньше 25% (без учета доп. задач), то все задание оценивается в 0 баллов.

Баллы за задание:

- Задача 1 - 7 баллов
- Задача 2 - 2 балла
- Задача 3* - 3 балла
- Задача 4 - 2 балла
- Задача 5* - 10 баллов
- Задача 6 - 5 баллов
- Задача 7 - 4 балла
- Задача 8* - 4 балла
- Задача 9* - 10 баллов

1. Линейная регрессия

Задача 1. По шаблону напишите класс, реализующий линейную регрессию. Интерфейс этого класса в некоторой степени соответствует классу `LinearRegression` (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) из библиотеки `sklearn`.

In [2]:
```
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt
import pandas
from pandas import DataFrame
import scipy.linalg as slin
import sklearn
```

In [3]:
```
class LinearRegression:
    def __init__(self):
        super()

    def fit(self, X, Y, alpha=0.95):
        ''' Обучение модели. Предполагается модель Y = X * theta + epsilon,
            где X --- регрессор, Y --- отклик,
            а epsilon имеет нормальное распределение с параметрами (0, sigma^2 * I_n).
            alpha --- уровень доверия для доверительного интервала.
        '''

        self.n, self.k = X.shape

        self.theta = np.linalg.inv(X.T @ X) @ X.T @ Y
        self.sigma_sq = ((Y - X @ self.theta).T @ (Y - X @ self.theta)) / (self.n - self.k)
        self.conf_int = np.empty(shape = (self.k, 2), dtype = float)
        t1 = sps.t.ppf((1 + alpha) / 2, df=self.n - self.k)
        t2 = sps.t.ppf((1 - alpha) / 2, df=self.n - self.k)
        d = (self.sigma_sq * np.linalg.inv(X.T @ X).diagonal()) ** (0.5)
        self.conf_int[:, 0] = self.theta.T - d * t1
        self.conf_int[:, 1] = self.theta.T - d * t2
        return self

    def summary(self):
        print('\nLinear regression on %d features and %d examples' % (self.k, self.n))
        print('Sigma: %.6f' % self.sigma_sq)
        print('\t\tLower\t\tEstimation\tUpper')
        for j in range(self.k):
            print('theta_%d:\t%.6f\t%.6f\t%.6f' % (j, self.conf_int[j, 0],
                                                   self.theta[j], self.conf_int[j, 1]))
        print('\n')

    def predict(self, X):
        ''' Возвращает предсказание отклика на новых объектах X. '''

        Y_pred = X @ self.theta
        return Y_pred
```

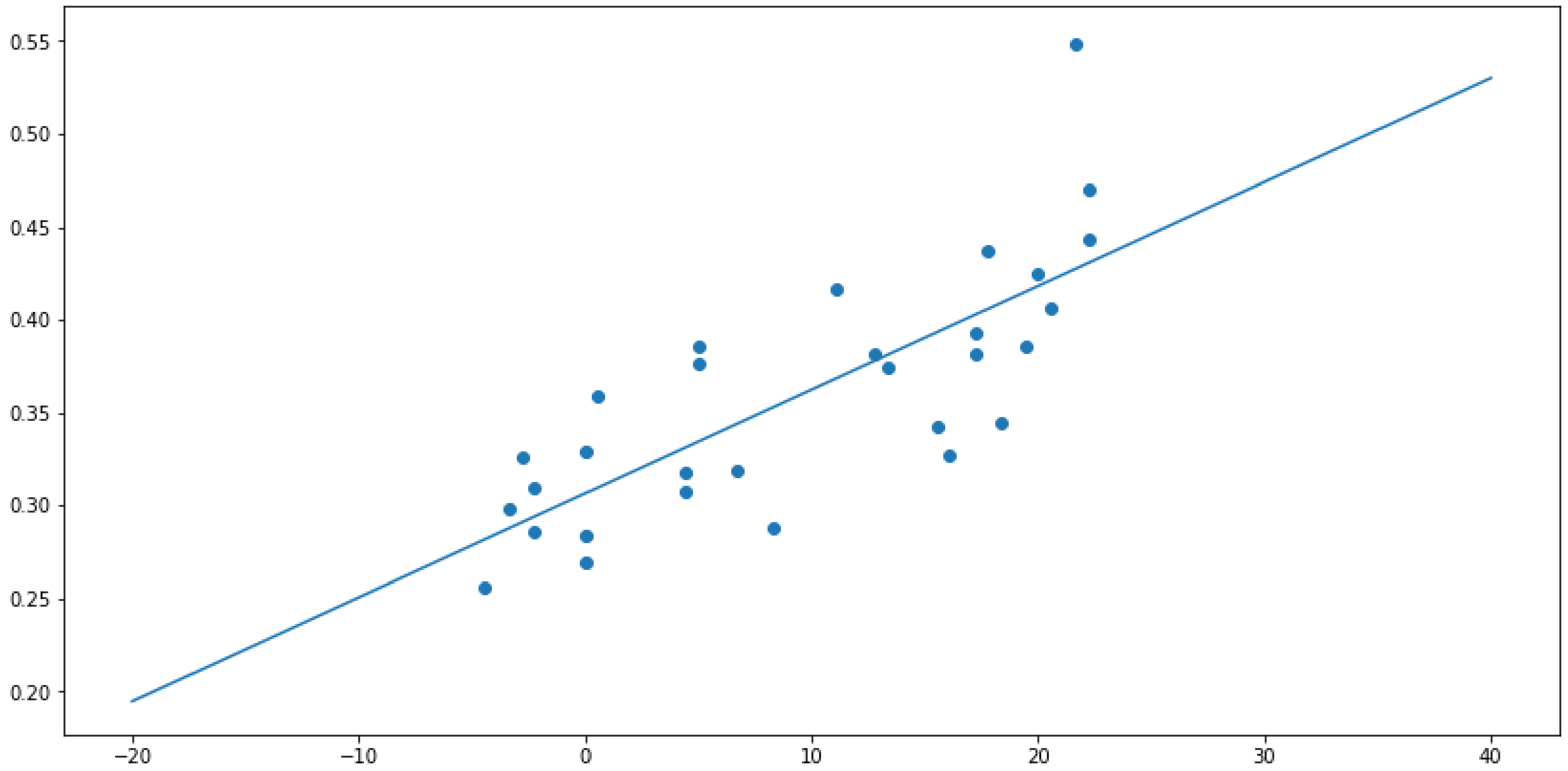
Загрузите данные о потреблении мороженного в зависимости от температуры воздуха и цены (файл `ice_cream.txt`). Примените реализованный выше класс линейной регрессии к этим данным предполагая, что модель имеет вид $ic = \theta_1 + \theta_2\ t$, где t --- температура воздуха (столбец `temp`), ic --- постребление мороженного в литрах на человека (столбец `IC`). Значения температуры предварительно переведите из Фаренгейта в Цельсий [(Фаренгейт — 32) / 1,8 = Цельсий].

К обученной модели примените фунцию `summary` и постройте график регрессии, то есть график прямой $ic = \widehat{\theta}_1 + \widehat{\theta}_2\ t$, где $\widehat{\theta}_1, \widehat{\theta}_2$ --- МНК-оценки коэффициентов. На график нанесите точки выборки. Убедитесь, что построейнный график совпадает с графиком из презентации с первой лекции, правда, с точностью до значений температура (она была неправильно переведена из Фаренгейта в Цельсий).

In [14]:
```
data = pandas.read_csv("ice_cream.txt", sep='\t')
ic = np.array(data["IC"])
t = np.array(data["temp"])
year = np.array(data["Year"])
income = np.array(data["income"])
lt = np.array(data["Lag-temp"])
price = np.array(data["price"])
t = (t - 32) / 1.8
ic = ic.reshape(ic.shape[0], 1)
year1 = [1 if value == 1 else 0 for value in year]
year2 = [1 if value == 2 else 0 for value in year]
```

In [5]:
```
intercept = np.ones(shape = t.shape)
X = np.empty(shape = (t.shape[0], 2))
X[:, 1] = t
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic)
model.summary()
x = np.linspace(-20, 40, 1000)
X1 = np.empty(shape = (x.shape[0], 2))
X1[:, 1] = x
X1[:, 0] = np.ones(shape= x.shape)
y = model.predict(X1)
plt.figure(figsize=(14, 7))
plt.plot(x, y)
plt.scatter(t, ic)
plt.show()

residuals_1 = (ic - model.predict(X)).reshape(30)  # для задачи №7
```

```
Linear regression on 2 features and 30 examples
Sigma: 0.001786
                Lower           Estimation      Upper
theta_0:        0.283276        0.306298        0.329319
theta_1:        0.003831        0.005593        0.007355
```



Вывод: построейнный график совпадает с графиком из презентации с первой лекции (с точностью до значений температуры)

Теперь учтите влияние года (столбец `Year`) для двух случаев:

- модель $ic = \theta_1 + \theta_2\ t + \theta_3 y_1 + \theta_4 y_2$, где $y_1 = I\{1\ \text{год}\}, y_2 = I\{2\ \text{год}\}$. Поясните, почему нельзя рассматривать одну переменную y --- номер года.
- для каждого года рассматривается своя линейная зависимость $ic = \theta_1 + \theta_2\ t$.

В каждом случае нарисуйте графики. Отличаются ли полученные результаты? От чего это зависит? Как зависит потребление мороженного от года?

Вывод: нельзя рассматривать одну переменную y --- номер года, так как это некоторая константа, которая для второго года будет умножаться на два, а для первого на один, то есть для второго всегда будет в два раза больше, что не есть хорошо. То есть эта переменная y отвечает не за числовой признак, а служит своеобразным флагом категории.

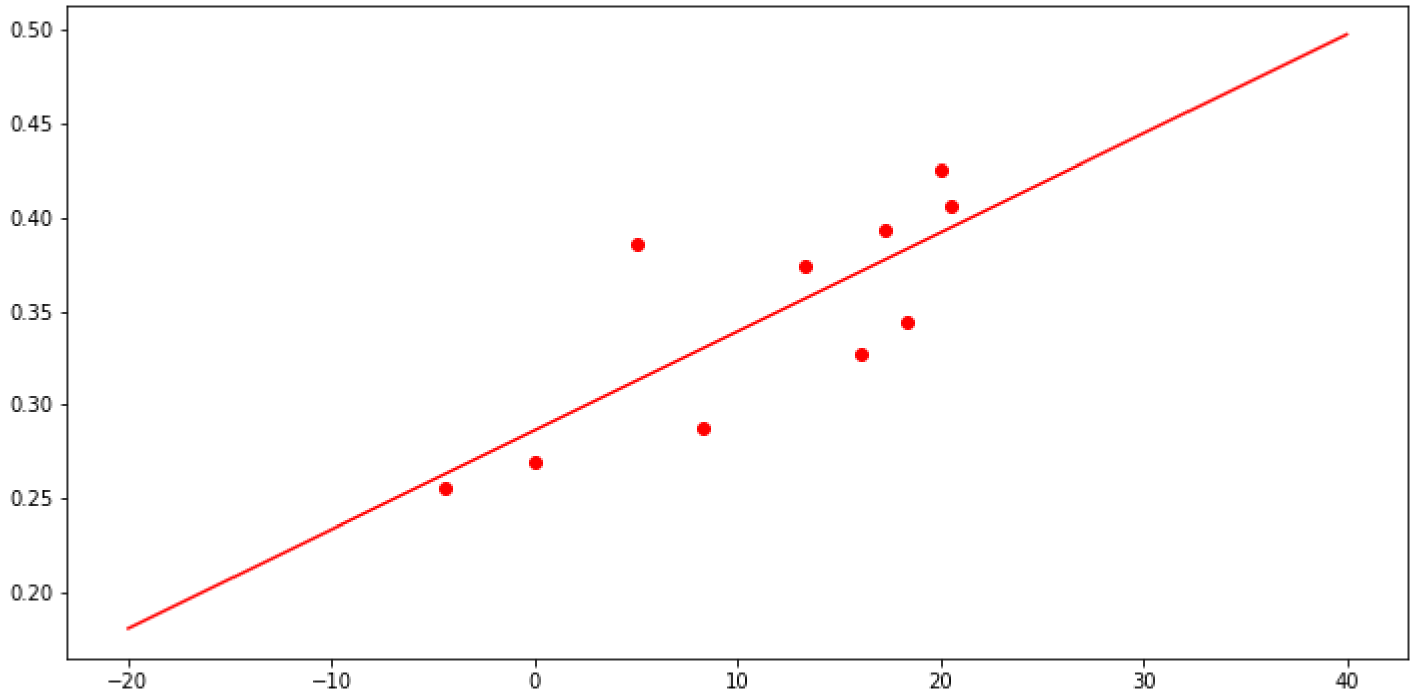
In [6]:
```
x = np.linspace(-20, 40, 1000)
```

In [7]:
```
# Общая модель
intercept = np.ones(shape = t.shape)
X = np.empty(shape = (t.shape[0], 4))
X[:, 3] = year2
X[:, 2] = year1
X[:, 1] = t
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic)
print('Общая модель')
model.summary()

residuals_2 = (ic - model.predict(X)).reshape(30)  # для задачи №7
```

```
Общая модель

Linear regression on 4 features and 30 examples
Sigma: 0.001016
                Lower           Estimation      Upper
theta_0:        0.251176        0.277050        0.302923
theta_1:        0.004741        0.006095        0.007449
theta_2:        -0.011237       0.016491        0.044218
theta_3:        0.041535        0.074307        0.107078
```

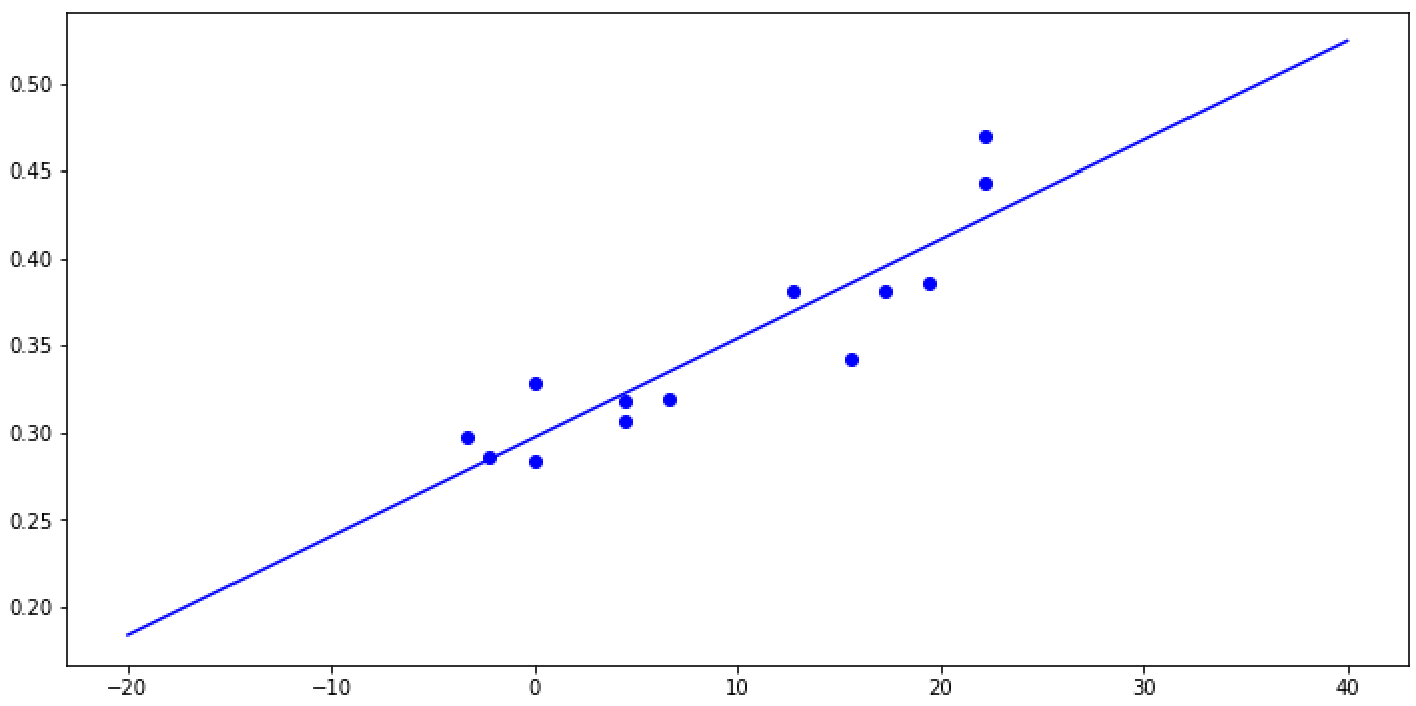
In [8]:
```
# Модель для нулевого года
ic0 = ic[0: 10]
t0 = t[0: 10]
ic0 = ic0.reshape(ic0.shape[0], 1)
intercept = np.ones(shape = t0.shape)
X = np.empty(shape = (t0.shape[0], 2))
X[:, 1] = t0
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic0)
print('Модель для нулевого года')
model.summary()
X1 = np.empty(shape = (x.shape[0], 2))
X1[:, 1] = x
X1[:, 0] = np.ones(shape= x.shape)
y = model.predict(X1)
plt.figure(figsize=(12, 6))
plt.plot(x, y, color='red')
plt.scatter(t0, ic0, color='red')
plt.show()

residuals_3 = (ic0 - model.predict(X)).reshape(10)  # для задачи №7
```

```
Модель для нулевого года

Linear regression on 2 features and 10 examples
Sigma: 0.001597
                Lower           Estimation      Upper
theta_0:        0.236963        0.286405        0.335846
theta_1:        0.001787        0.005277        0.008767
```



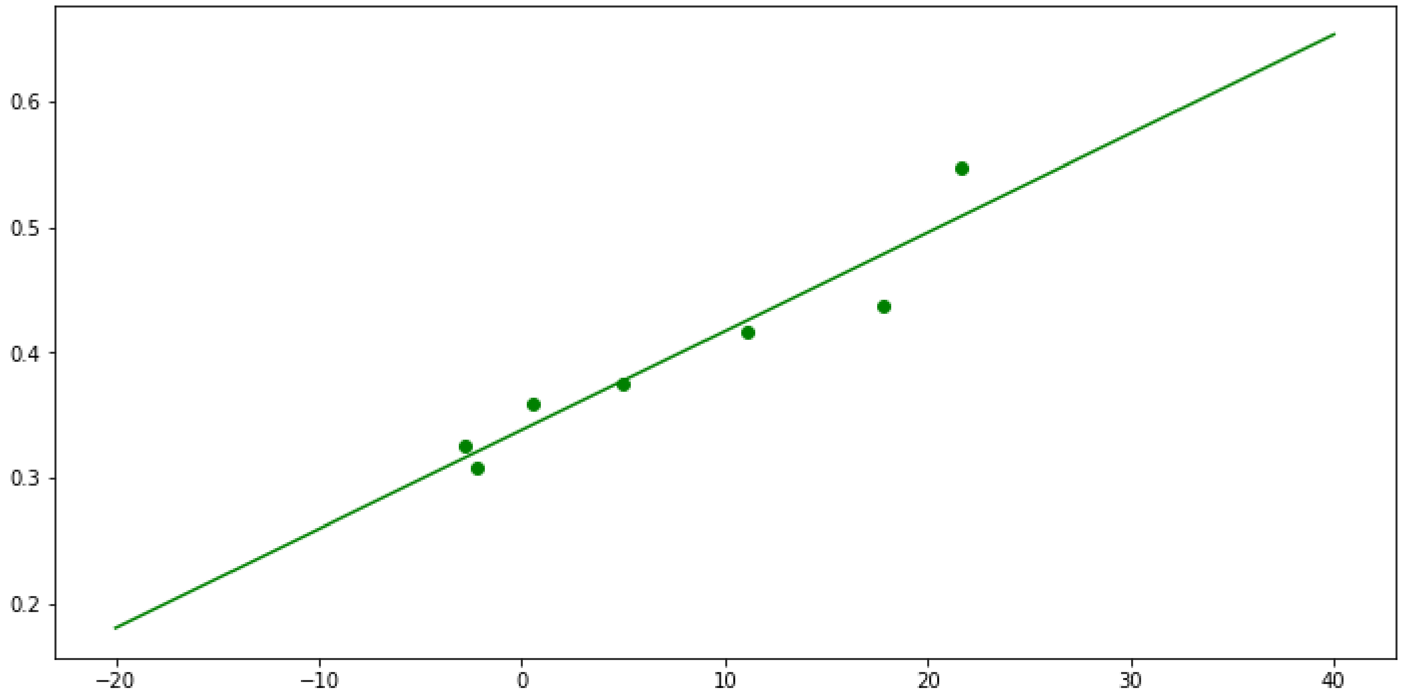
In [9]:
```
# Модель для первого года
ic1 = ic[10: 23]
t1 = t[10: 23]
ic1 = ic1.reshape(ic1.shape[0], 1)
intercept = np.ones(shape = t1.shape)
X = np.empty(shape = (t1.shape[0], 2))
X[:, 1] = t1
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic1)
print('Модель для первого года')
model.summary()
X1 = np.empty(shape = (x.shape[0], 2))
X1[:, 1] = x
X1[:, 0] = np.ones(shape= x.shape)
y = model.predict(X1)
plt.figure(figsize=(12, 6))
plt.plot(x, y, color='blue')
plt.scatter(t1, ic1, color='blue')
plt.show()

residuals_4 = (ic1 - model.predict(X)).reshape(13)  # для задачи №7
```

```
Модель для первого года

Linear regression on 2 features and 13 examples
Sigma: 0.000667
                Lower           Estimation      Upper
theta_0:        0.274993        0.297426        0.319859
theta_1:        0.003935        0.005672        0.007409
```



In [10]:
```
# Модель для второго года
ic2 = ic[23:]
t2 = t[23:]
ic2 = ic2.reshape(ic2.shape[0], 1)
intercept = np.ones(shape = t2.shape)
X = np.empty(shape = (t2.shape[0], 2))
X[:, 1] = t2
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic2)
print('Модель для второго года')
model.summary()
X1 = np.empty(shape = (x.shape[0], 2))
X1[:, 1] = x
X1[:, 0] = np.ones(shape= x.shape)
y = model.predict(X1)
plt.figure(figsize=(12, 6))
plt.plot(x, y, color='green')
plt.scatter(t2, ic2, color='green')
plt.show()

residuals_5 = (ic2 - model.predict(X)).reshape(7)  # для задачи №7
```

```
Модель для второго года

Linear regression on 2 features and 7 examples
Sigma: 0.000766
                Lower           Estimation      Upper
theta_0:        0.303805        0.338346        0.372886
theta_1:        0.004907        0.007877        0.010846
```



Вывод: Из построенных графиков видно, что с течением времени потребление мороженого растет. Также обратим внимание, что для второго года точки лежат очень близко к прямой, то есть нам удалось хорошо построить модель. Для первого года чуть хуже, для нулевого --- еще хуже.

Наконец, обучите модель на предсказание потребления мороженного в зависимости от всех переменных. Не забудьте, что для года нужно ввести две переменных. Для полученной модели выведите `summary`.

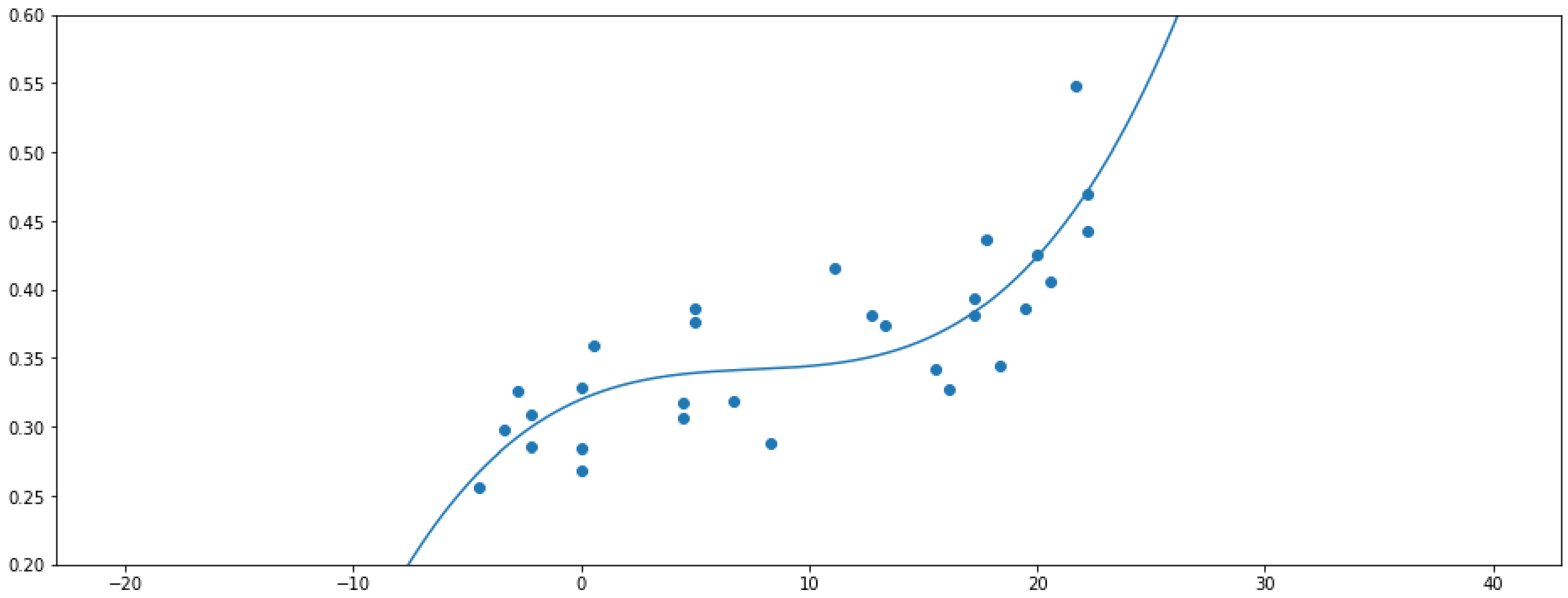
In [11]:
```
intercept = np.ones(shape = t.shape)
X = np.empty(shape = (t.shape[0], 7))
X[:, 6] = lt
X[:, 5] = income
X[:, 4] = price
X[:, 3] = year2
X[:, 2] = year1
X[:, 1] = t
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic)
model.summary()

residuals_6 = (ic - model.predict(X)).reshape(30)  # для задачи №7
```

```
Linear regression on 7 features and 30 examples
Sigma: 0.001024
                Lower           Estimation      Upper
theta_0:        0.107657        0.717753        1.327849
theta_1:        0.003801        0.005654        0.007507
theta_2:        -0.000852       0.038141        0.077134
theta_3:        0.045224        0.117733        0.190242
theta_4:        -2.467091       -0.659296       1.148500
theta_5:        -0.007774       -0.003231       0.001311
theta_6:        -0.000886       -0.000024       0.000838
```

Но это еще не все. Постройте теперь линейную регрессию для модели $ic = \theta_1 + \theta_2\ t + \theta_3\ t^2 + \theta_4\ t^3$. Выведите для нее summary и постройте график предсказания, то есть график кривой $ic = \widehat{\theta}_1 + \widehat{\theta}_2\ t + \widehat{\theta}_3\ t^2 + \widehat{\theta}_4\ t^3$. Хорошие ли получаются результаты?

In [13]:
```
t_2 = t ** 2
t_3 = t ** 3
intercept = np.ones(shape = t.shape)
X = np.empty(shape = (t.shape[0], 4))
X[:, 3] = t_3
X[:, 2] = t_2
X[:, 1] = t
X[:, 0] = intercept
model = LinearRegression()
model = model.fit(X, ic)
model.summary()
X1 = np.empty(shape = (x.shape[0], 4))
X1[:, 3] = x ** 3
X1[:, 2] = x ** 2
X1[:, 1] = x
X1[:, 0] = np.ones(shape= x.shape)
y = model.predict(X1)
plt.figure(figsize=(16,6))
plt.plot(x, y)
plt.scatter(t, ic)
plt.ylim(0.2, 0.6)
plt.show()

residuals_7 = (ic - model.predict(X)).reshape(30)  # для задачи №7
```

```
Linear regression on 4 features and 30 examples
Sigma: 0.001529
                Lower           Estimation      Upper
theta_0:        0.295294        0.319902        0.344510
theta_1:        0.000388        0.007200        0.014013
theta_2:        -0.001861       -0.000855       0.000152
theta_3:        0.000002        0.000038        0.000073
```



Вывод: Исходя из графика можно сделать вывод, что результаты получились хорошие. Точки лежат довольно близко к построенной кривой.

Чтобы понять, почему так происходит, выведите значения матрицы $(X^T X)^{-1}$ для данной матрицы и посчитайте для нее индекс обусловленности $\sqrt{\lambda_{max} / \lambda_{min}}$, где $\lambda_{max}, \lambda_{min}$ --- максимальный и минимальный собственные значения матрицы $X^T X$. Собственные значения можно посчитать функцией `scipy.linalg.eigvals` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigvals.html).

Прокомментируйте полученные результаты. Помочь в этом может следующая статья (https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%BE%D0%B1%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B

In [56]:
```
eigvals = slin.eigvals(np.linalg.inv(X.T @ X))
eigvals
```

Out[56]:
```
array([  9.39587724e-02+0.j,   7.10180670e-03+0.j,   2.97693228e-05+0.j,
         1.41790261e-09+0.j])
```

In [57]:
```
index = (eigvals.max() / eigvals.min()) ** (0.5)
index
```

Out[57]:
```
(8140.3947488960366+0j)
```

Вывод: мы получили, что инедкс обусловленности очень большой (из приведенной статьи: идеальный = 1). Чем больше индекс обусловленности, тем больше дисперсия параметров, а чем больше дисперсия, тем шире разброс. Также, так как индекс обусловленности > 30, то имеет место мультиколлинеарность (Это значит, что "всё плохо").

Задача 2. В данной задаче нужно реализовать функцию отбора признаков для линейной регрессии. Иначе говоря, пусть есть модель $y = \theta_1 x_1 + \ldots + \theta_k x_k$. Нужно определить, какие θ_j нужно положить равными нулю, чтобы качество полученной модели было максимальным.

Для этого имеющиеся данные нужно случайно разделить на две части --- обучение и тест (train и test). На первой части нужно обучить модель регресии, взяв некоторые из признаков, то есть рассмотреть модель $y = \theta_{j_1} x_{j_1} + \ldots + \theta_{j_s} x_{j_s}$. По второй части нужно посчитать ее качество --- среднеквадратичное отклонение (mean squared error) предсказания от истинного значения отклика, то есть величину

$$MSE = \sum_{i \in test} \left(\widehat{y}(x_i) - Y_i\right)^2,$$

где $x_i = (x_{i,1}, \ldots, x_{i,k})$, Y_i --- отклик на объекте x_i, а $\widehat{y}(x)$ --- оценка отклика на объекте x.

Если k невелико, то подобным образом можно перебрать все поднаборы признаков и выбрать наилучший по значению MSE.

Для выполнения задания воспользуйтесь следующими функциями:

- `sklearn.linear_model.LinearRegression` (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) --- реализация линейной регрессии. В данной реализации свободный параметр θ_1 по умолчанию автоматически включается в модель. Отключить это можно с помощью `fit_intercept=False`, но это не нужно. *В данной задаче требуется, чтобы вы воспользовались готовой реализацией линейной регрессии, а не своей. Ведь на практике важно уметь применять готовые реализации, а не писать их самостоятельно.*
- `sklearn.cross_validation.train_test_split` (http://scikit-learn.org/0.16/modules/generated/sklearn.cross_validation.train_test_split.html) --- функция разбиения данных на train и test. Установите параметр `test_size=0.3`.
- `sklearn.metrics.mean_squared_error` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html) --- реализация MSE.

Для перебора реализуйте функцию.

In [15]:
```
from sklearn.datasets import load_boston
from sklearn.cross_validation import train_test_split
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
```

In [16]:
```
def best_features(X_train, X_test, Y_train, Y_test):
    mses = []
    k = X_train.shape[1]

    for j in range(1, 2 ** k):  # номер набора признаков
        mask = np.array([j & (1 << s) for s in range(k)], dtype=bool)
        features_numbers = np.arange(k)[mask]  # набор признаков

        X_train_temp = X_train[:, features_numbers]
        X_test_temp = X_test[:, features_numbers]
        model = linear_model.LinearRegression()
        model = model.fit(X_train_temp, Y_train)
        mse = mean_squared_error(Y_test, model.predict(X_test_temp))
        mses.append(mse)

    # Печать 10 лучших наборов
    print('mse\t features')
    mses = np.array(mses)
    best_numbres = np.argsort(mses)[:10]
    for j in best_numbres:
        mask = np.array([j & (1 << s) for s in range(k)], dtype=bool)
        features_numbers = np.arange(k)[mask]
        print('%.3f\t' % mses[j], features_numbers)
```

Примените реализованный отбор признаков к датасетам

- Yacht Hydrodynamics (http://archive.ics.uci.edu/ml/datasets/Yacht+Hydrodynamics) --- для парусных яхт нужно оценить остаточное сопротивление на единицу массы смещения (последний столбец) в зависимости от различных характеристик яхты.
- Boston Housing Prices (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston) --- цены на дома в Бостоне в зависимости от ряда особенностей.

In [19]:
```
data = load_boston().data
X = data[:,range(0, 12)]
Y = data[:, 12]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
print("Boston_prices")
best_features(X_train, X_test, Y_train, Y_test)
```

```
Boston_prices
mse      features
18.711   [ 2  3  4  5  6  7  8 11]
18.734   [ 2  3  4  5  6  7  8 10 11]
18.743   [ 2  3  4  5  6  7  9 11]
18.785   [ 1  2  3  4  5  6  7  8 11]
18.792   [ 2  3  4  5  6  7  9 10 11]
18.796   [ 3  4  5  6  7  9 11]
18.816   [ 2  3  4  5  6  7  8  9 11]
18.821   [ 2  3  5  6  7  8 11]
18.822   [ 2  3  4  5  6  7 10 11]
18.832   [ 2  3  5  6  7  8 10 11]
```

Вывод: заметим, что самый часто встречающийся признак - признак №5.

In [18]:
```
data = pandas.read_csv("yacht.txt", header=None, delim_whitespace=True)
data = np.array(data)
X = data[:,range(0, 6)]
Y = data[:, 6]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
print("Yachts")
best_features(X_train, X_test, Y_train, Y_test)
```

```
Yachts
mse      features
77.211   [0 5]
77.240   [0 4 5]
77.360   [1 5]
77.395   [1 4 5]
77.567   [0 1 2 3 4]
77.605   [0 1 2 3 5]
77.606   [0 2 5]
77.713   [5]
77.743   [1 2 5]
77.756   [4 5]
```

Вывод: здесь аналогично самый часто встречающийся признак - признак №5.

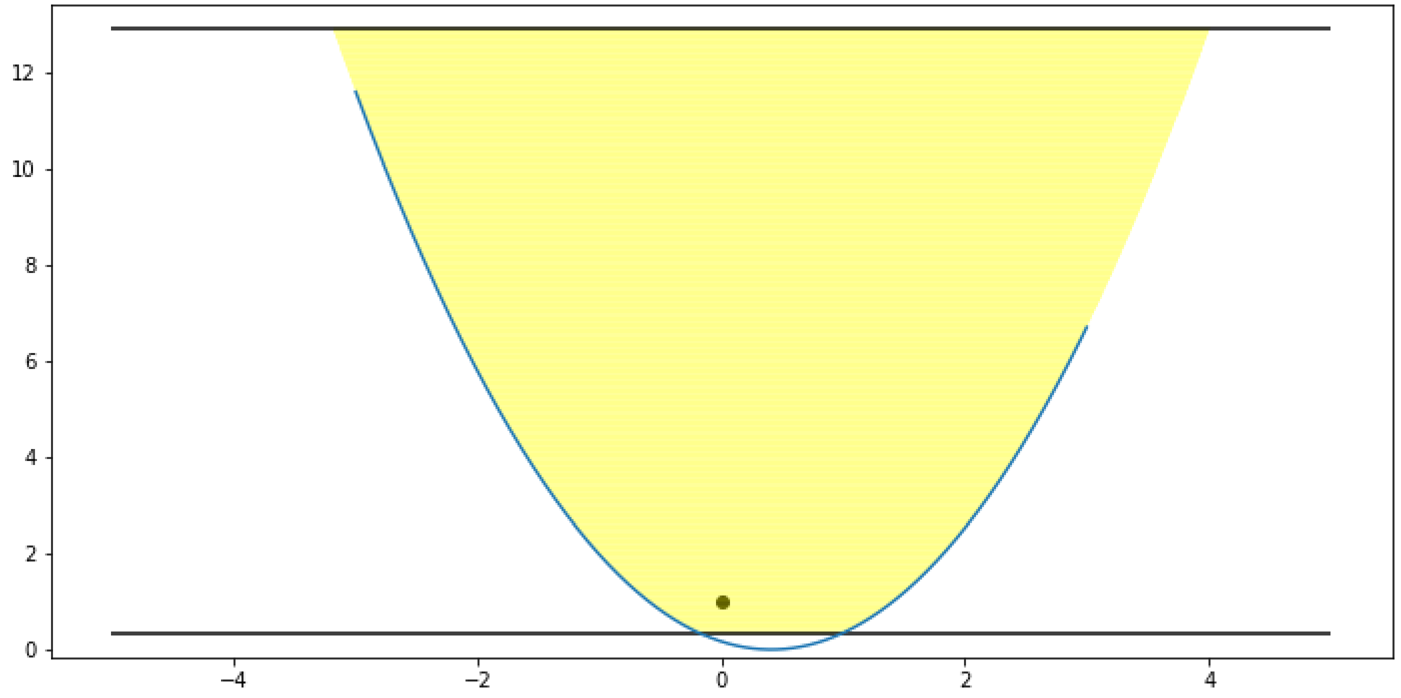
Задача 3*. Загрузите датасет (http://people.sc.fsu.edu/~jburkardt/datasets/regression/x01.txt), в котором показана зависимость веса мозга от веса туловища для некоторых видов млекопитающих. Задача состоит в том, чтобы подобрать по этим данным хорошую модель регрессии. Для этого, можно попробовать взять некоторые функции от значения веса туловища, например, степенную, показательную, логарифмическую. Можно также сделать преобразование значений веса мозга, например, прологарифмировать. Кроме того, можно разбить значения веса туловища на несколько частей и на каждой части строить свою модель линейной регрессии.

Задача 4. Пусть $X_1, \ldots, X_n$ --- выборка из распределения $\mathcal{N}(a, \sigma^2)$. Постройте точную доверительную область для параметра $\theta = (a, \sigma^2)$ уровня доверия $\alpha = 0.95$ для сгенерированной выборки размера $n \in \{5, 20, 50\}$ из стандартного нормального распределения. Какой вывод можно сделать?

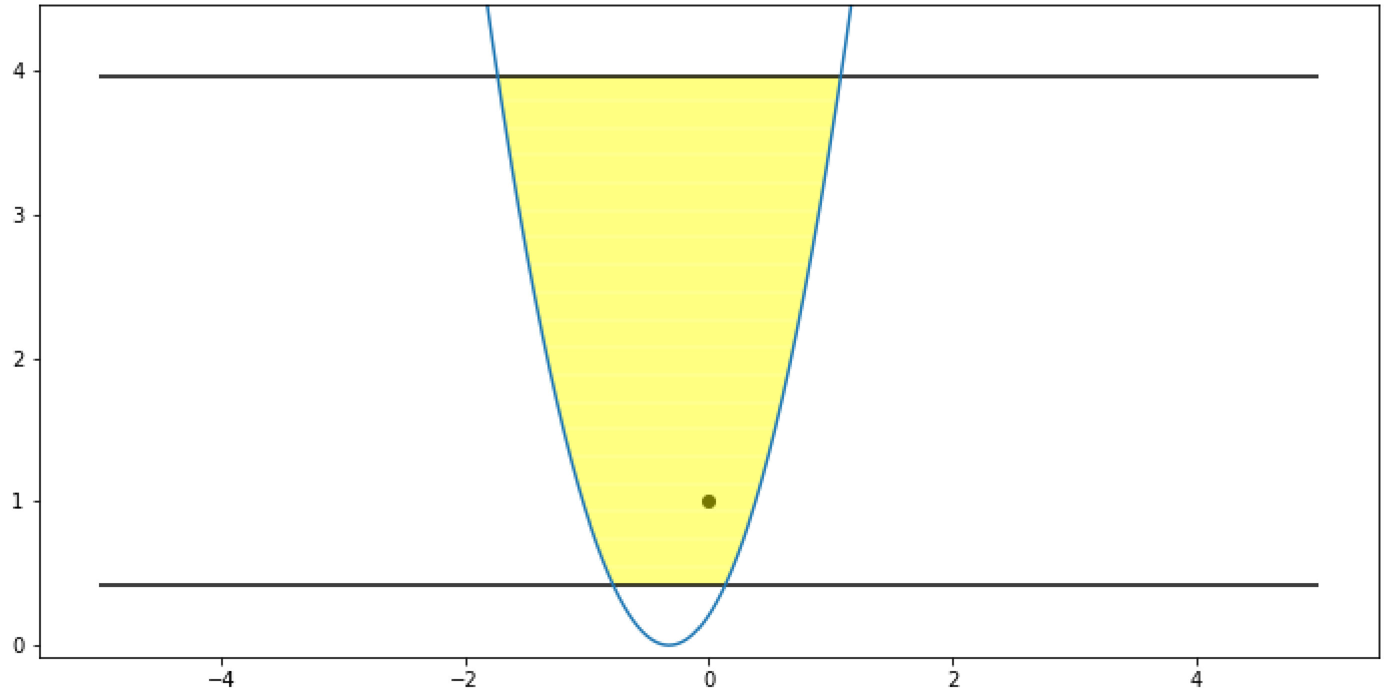
Точную доверительную область для параметра $\theta = (a, \sigma^2)$ уровня доверия $\alpha = 0.95$ знаем из домашнего задания.

In [24]:
```
def draw_conf(X, alpha = 0.95 ** 0.5):
    n = X.size
    mean = X.mean()
    S = ((X - mean) ** 2 / n).sum()
    z1 = sps.chi2.ppf((1 + alpha) / 2, df=n - 1)
    z2 = sps.chi2.ppf((1 - alpha) / 2, df=n - 1)
    x1 = sps.norm.ppf((1 + alpha) / 2)
    theta2 = (n * S / z2)
    theta1 = (n * S / z1)
    x = np.linspace(-3, 3, 1000)
    y = ((x - mean) * (n ** (0.5)) / x1) ** 2
    plt.figure(figsize=(12,6))
    plt.plot(x, y)
    lines = np.linspace(theta1, theta2, 1000)
    plt.scatter(0, 1, color = 'black')
    for temp in lines:
        v = (temp / n) ** 0.5 * x1
        plt.hlines(temp, -v + mean, v + mean, color = 'yellow', alpha = 0.1)
    plt.hlines(theta1, xmax = 5, xmin = -5)
    plt.hlines(theta2, xmax = 5, xmin = -5)
    plt.ylim((theta1 - 0.5, theta2 + 0.5))
    plt.show()
```

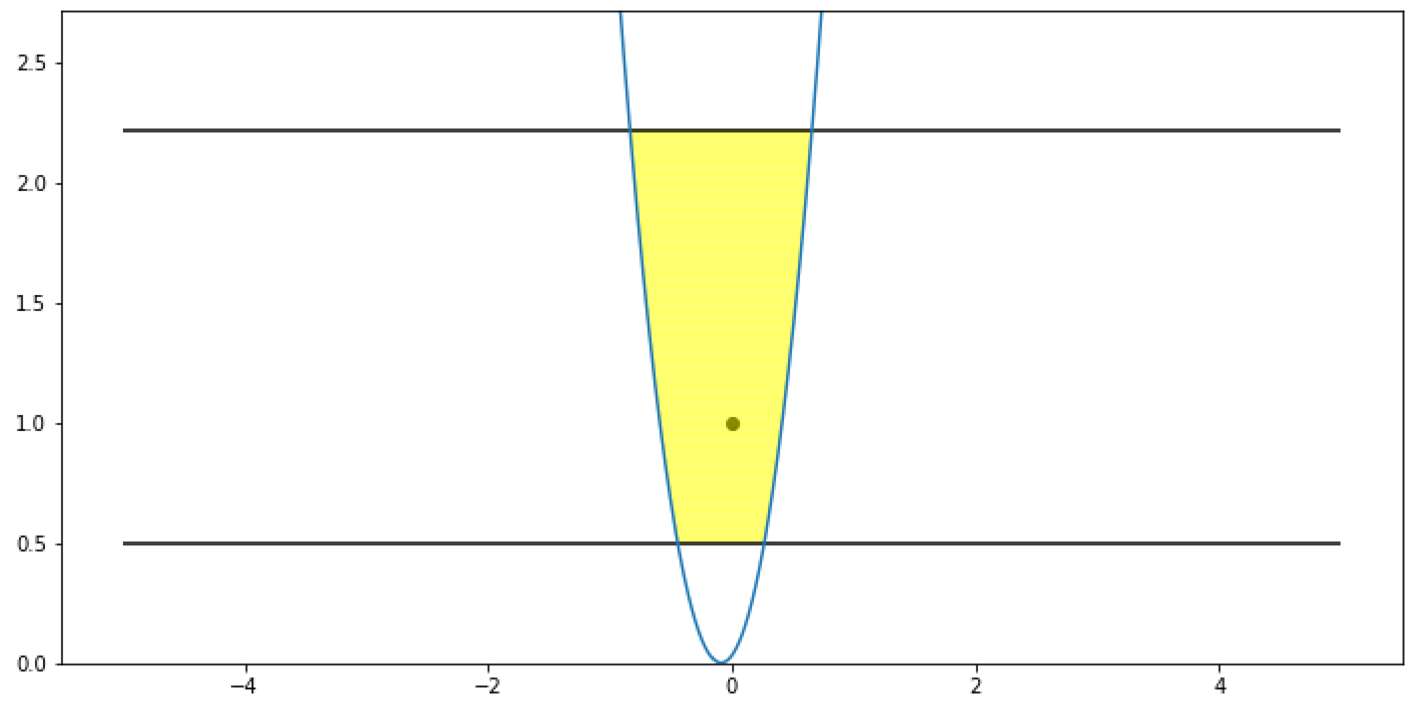
In [25]:
```
sample = sps.norm.rvs(size = 5, loc = 0)
draw_conf(sample)
```



In [26]:
```
sample = sps.norm.rvs(size = 10, loc = 0)
draw_conf(sample)
```



In [27]:
```
sample = sps.norm.rvs(size = 20, loc = 0)
draw_conf(sample)
```



Вывод: Несколько раз сгенерировав выборку, убедились в том, что доверительная область оценивает параметр (обозначили его точкой на графике) с высокой точностью. С ростом выборки истинное значение всё точнее попадает в построенную область.

Задача 5*. Пусть дана линейная гауссовская модель $Y = X\theta + \varepsilon$, где $\varepsilon \sim \mathcal{N}(0, \beta^{-1}I_n)$. Пусть θ имеет априорное распределение $\mathcal{N}(0, \alpha^{-1}I_k)$. Такая постановка задачи соответствует Ridge-регрессии. Оценкой параметров будет математическое ожидание по апостериорному распределению, аналогично можно получить доверительный интервал. Кроме того, с помощью апостериорного распределения можно получить доверительный интервал для отклика на новом объекте, а не только точечную оценку.

Реализуйте класс RidgeRegression подобно классу LinearRegression, но добавьте в него так же возможность получения доверительного интервала для отклика на новом объекте. Примените модель к некоторых датасетам, которые рассматривались в предыдущих задачах. Нарисуйте графики оценки отклика на новом объекте и доверительные интервалы для него.

2. Проверка статистических гипотез

Задача 6. Существует примета, что если перед вам дорогу перебегает черный кот, то скоро случится неудача. Вы же уже достаточно хорошо знаете статистику и хотите проверить данную примету. Сформулируем задачу на математическом языке. Пусть $X_1, \dots, X_n \sim Bern(p)$ --- проведенные наблюдения, где $X_i = 1$, если в i-м испытании случилась неудача после того, как черный кот перебежал дорогу, а p --- неизвестная вероятность такого события. Нужно проверить гипотезу $H_0 : p = 1/2$ (отсутствие связи между черным котом и неудачей) против альтернативы $H_1 : p > 1/2$ (неудача происходит чаще если черный кот перебегает дорогу).

Известно, что $S = \{T(X) > c_\alpha\}$, где $T(X) = \sum X_i$, является равномерно наиболее мощным критерием для данной задачи. Чему при этом равно c_α? При этом p-value в данной задаче определяется как $p(t) = \mathsf{P}_{0.5}(T(X) > t)$, где $t = \sum x_i$ --- реализация статистики $T(X)$.

Для начала проверьте, что критерий работает. Возьмите несколько значений n и реализаций статистики $T(X)$. В каждом случае найдите значение c_α и p-value. Оформите это в виде таблицы.

Пользуйтесь функциями из `scipy.stats`, про которые подробно написано в файле `python_5`. Внимательно проверьте правильность строгих и нестрогих знаков.

Возьмем уровень доверия $\alpha = 0.05$.

$P(T(X) > c_\alpha) <= \alpha$

$P(T(X) <= c_\alpha) <= 1 - \alpha$

$F(c_\alpha) <= 1 - \alpha$, где F(x) - функция распределения биномиального распределения для p=0,5.

Тогда c_α --- квантиль уровня $(1 - \alpha)$ биномиального распределения.

p-value = P(T(X) > t)

P(T(X) <= t) = 1 - p-value

Тогда p-value = 1 - P(T(X) <= t)

In [54]:
```
N = 10000
sample = sps.bernoulli.rvs(size=N, p=0.5)
t = sample.cumsum()
sample_size = np.arange(1, N + 1)
c_alpha = sps.binom.ppf(q=0.95, n=sample_size, p=0.5)
p_value = sps.binom.sf(t, n=sample_size, p=0.5)
```

In [53]:
```
num = np.array([10, 50, 100, 500, 1000, 10000])
data = {"N": num, "$C_{alpha}$": c_alpha[num - 1], "p-value": p_value[num - 1] }
table = DataFrame(data=data)
table
```

Out[53]:

	C_{alpha}	N	p-value
0	8.0	10	0.623047
1	31.0	50	0.443862
2	58.0	100	0.617823
3	268.0	500	0.776435
4	526.0	1000	0.993778
5	5082.0	10000	0.181412

Вывод: Для нескольких рассмотренных значений n и реализаций статистики $T(X)$ получаем p-value >= 0,05. Значит, критерий работает, гипотезу не отвергаем.

Для каких истинных значений p с точки зрения практики можно считать, что связь между черным котом и неудачей есть? Теперь сгенерируйте 10 выборок для двух случаев: 1). $n = 5, p = 0.75$; 2). $n = 10^5, p = 0.51$. В каждом случае в виде таблицы выведите реализацию статистики $T(X)$, соответствующее p-value и 0/1 - отвергается ли H_0 (выводите 1, если отвергается). Какие выводы можно сделать?

1) $n = 5, p = 0.75$

In [110]:
```python
sample = sps.bernoulli.rvs(size=(10, 5), p=0.75)
t = sample.sum(axis=1)
c_alpha = sps.binom.ppf(q=0.95, n=5, p=0.5)
p_value = sps.binom.sf(t, n=5, p=0.5)
res = [1 if t_i > c_alpha else 0 for t_i in t]
data = {'T(X)': t, 'p-value': p_value, "$H_0$": res}
table = DataFrame(data=data)
table
```

Out[110]:

	H_0	T(X)	p-value
0	0	4	0.03125
1	0	3	0.18750
2	0	3	0.18750
3	0	4	0.03125
4	0	4	0.03125
5	1	5	0.00000
6	0	4	0.03125
7	1	5	0.00000
8	1	5	0.00000
9	0	3	0.18750

2) $n = 10^5, p = 0.51$

In [71]:
```python
sample = sps.bernoulli.rvs(size=(10, 10 ** 5), p=0.51)
t = sample.sum(axis=1)
c_aplha = sps.binom.ppf(q=0.95, n=(10 ** 5), p=0.5)
p_value = sps.binom.sf(t, n=(10 ** 5), p=0.5)
res = [1 if t_i > c_alpha else 0 for t_i in t]
data = {"$T(X)$": t, "p_value": p_value, "$H_0$": res}
table = DataFrame(data=data)
table
```

Out[71]:

	H_0	$T(X)$	p_value
0	1	51240	2.147467e-15
1	1	51058	1.079935e-11
2	1	50906	4.922968e-09
3	1	50739	1.454873e-06
4	1	50951	8.829750e-10
5	1	51234	2.903891e-15
6	1	50842	4.949738e-08
7	1	50712	3.298108e-06
8	1	51072	5.871440e-12
9	1	50993	1.653603e-10

Вывод: Видим, что для достаточно большого размера выборки (n = 10^5) даже такое незначительное изменение истинной вероятности (на 0,01) влияет, и тем самым гипотезу отвергаем в 10 из 10 случаях. А для маленького размера выборки (n=5) даже с таким значимым изменением вероятности p на 0,25 мы не отвергли гипотезу в больше половины случаев. То есть определять есть ли связь между черным котом и неудачей с точки зрения практики для различных истинных p можем исходя из размера выборки.

Возникает задача подбора оптимального размера выборки.

Для этого сначала зафиксируйте значение $p^* > 1/2$, которое будет обладать следующим свойством. Если истинное $p > p^*$, то такое отклонение от $1/2$ с практической точки зрения признается существенным, то есть действительно чаще случается неудача после того, как черный кот перебегает дорогу. В противном случае отклонение с практической точки зрения признается несущественным.

Теперь для некоторых n постройте графики функции мощности критерия при $1/2 < p < 1$ и уровне значимости 0.05. Выберите такое n^*, для которого функция мощности дает значение 0.8 при p^*. Для выбранного n^* проведите эксперимент, аналогичный проведенным ранее экспериментам, сгенерировав выборки для следующих истинных значений p: 1). $1/2 < p < p^*$; 2). $p > p^*$. Сделайте вывод.

Из условия: уровень доверия $\alpha = 0.05$.

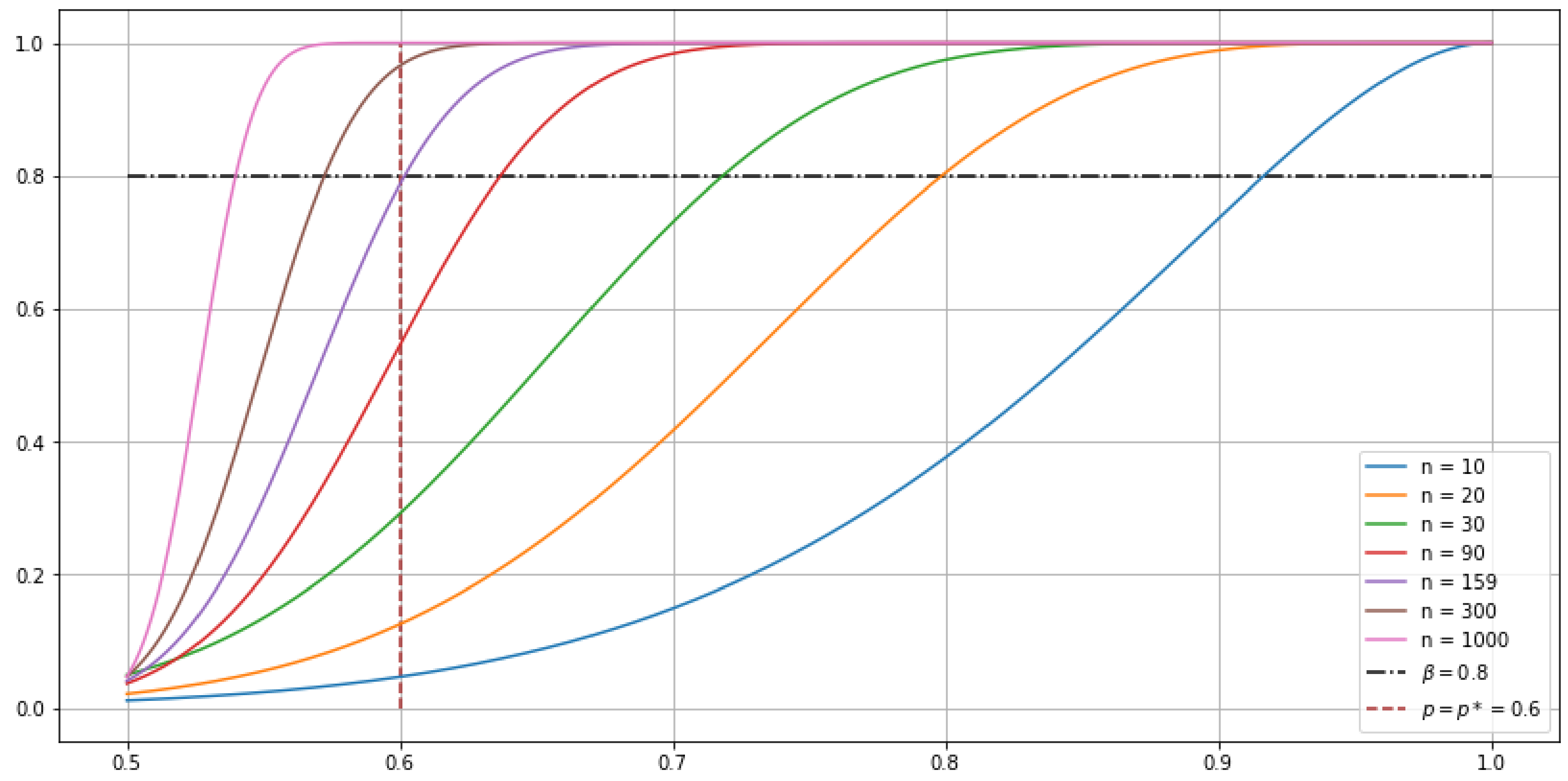
Аналогично предыдущим рассуждениям: c_α --- квантиль уровня $(1 - \alpha)$ биномиального распределения.

beta = P(T(X) > c_α)

P(T(X) <= c_α) = 1 - beta

Тогда beta = 1 - P(T(X) <= c_α)

In [155]:
```
p_star = 0.6  # зафиксировали значение p* из условия

n = np.array(object=[10, 20, 30, 90, 159, 300, 1000])
p = np.linspace(0.5, 1, 1000)
plt.figure(figsize=(14, 7))

for i in range(n.size):
    c_alpha = sps.binom.ppf(q=0.95, n=n[i], p=0.5)
    beta = sps.binom.sf(c_alpha, n=n[i], p=p)
    plt.plot(p, beta, label='n = {}'.format(n[i]))
plt.hlines(0.8, 0.5, 1, linestyles='-.', label='$\\beta=0.8$')
plt.vlines(0.6, 0, 1, linestyles='--', label='$p=p*=0.6$', colors='brown')
plt.grid()
plt.legend()
plt.show()
```



Из графика выбрали $n^* = 159$.

In [158]:
```
n_star = 159
```

1) $1/2 < p < p^*$

In [162]:
```
sample = sps.bernoulli.rvs(size=(10, n_star), p=0.55)
t = sample.sum(axis=1)
c_alpha = sps.binom.ppf(q=0.95, n=n_star, p=0.5)
p_value = sps.binom.sf(t, n=n_star, p=0.5)
res = [1 if t_i > c_alpha else 0 for t_i in t]
data = {'T(X)': t, 'p-value': p_value, "$H_0$": res}
table = DataFrame(data=data)
table
```

Out[162]:

	H_0	T(X)	p-value
0	1	92	0.019439
1	0	84	0.213920
2	0	88	0.076607
3	1	91	0.028328
4	0	83	0.262962
5	0	89	0.056215
6	0	81	0.375595
7	0	86	0.133413
8	0	77	0.624405
9	0	86	0.133413

2) $p > p^*$

In [166]:
```
sample = sps.bernoulli.rvs(size=(10, n_star), p=0.69)
t = sample.sum(axis=1)
c_alpha = sps.binom.ppf(q=0.95, n=n_star, p=0.5)
p_value = sps.binom.sf(t, n=n_star, p=0.5)
res = [1 if t_i > c_alpha else 0 for t_i in t]
data = {'T(X)': t, 'p-value': p_value, "$H_0$": res}
table = DataFrame(data=data)
table
```

Out[166]:

	H_0	T(X)	p-value
0	1	105	1.592098e-05
1	1	113	2.151821e-08
2	1	107	3.615800e-06
3	1	109	7.349752e-07
4	1	107	3.615800e-06
5	1	115	3.079297e-09
6	1	108	1.653238e-06
7	1	101	2.236738e-04
8	1	116	1.112215e-09
9	1	103	6.291268e-05

Вывод: Мы взяли $p^* = 0.6$. Для него подобрали из графика подходящий с точки зрения мощности (beta = 0.8) размер выборки n = 159. Провели две серии экспериментов. Для первого случая, действительно, убедились в том, что гипотеза H_0 почти никогда не отвергается, то есть такие значения $p: 1/2 < p < p^*$ не отличимы на практике при размере выборки $n = 159$ (то есть отклонение с практической точки зрения признается несущественным). А для второго случая, если истинное $p > p*$, то наоборот: H_0 почти всегда отвергаем, то есть такое отклонение от 1/2 с практической точки зрения признается существенным.

Справка для выполнения следующих задач

Критерий согласия хи-квадрат

`scipy.stats.chisquare` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html#scipy.stats.chisquare)`(f_obs, f_exp=None, ddof=0)`

`f_obs` --- число элементов выборки, попавших в каждый из интервалов

`f_exp` --- ожидаемое число элементов выборки (по умолчанию равномерное)

`ddof` --- поправка на число степеней свободы. Статистика асимптотически будет иметь распределение хи-квадрат с числом степеней свободы $k - 1 - ddof$, где k --- число интервалов.

Возвращает значение статистики критерия и соответствующее p-value.

Критерий согласия Колмогорова

`scipy.stats.kstest` (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html#scipy.stats.kstest)`(rvs, cdf, args=())`

`rvs` --- выборка

`cdf` --- функция распределения (сама функция или ее название)

`args` --- параметры распределения

Возвращает значение статистики критерия и соответствующее p-value.

Задача 7.

- Проверьте, что ваша выборка значений скорости ветра из задания 2 действительно согласуется с распределением Вейбулла.
- Проверьте, что при больших n распределение статистики из задач 3 и 4 задания 2 действительно хорошо приближают предельное распределение.
- Проверьте, что остатки в регрессии из задач выше нормальны.
- Подберите класс распределений для выборки количества друзей из задания 1.

Использовать можно два описанных выше критерия, либо любой другой критерий, если будет обоснована необходимость его применения в данной задаче, а так же будет приведено краткое описание критерия. Уровень значимости взять равным 0.05.

Из задания 2 мы знаем, что для данной выборки параметры распределения Вейбулла:

c=2.10701, scale=3.08161

In [5]:
```
wind_speed = np.array([
    2.2, 3.3, 1.1, 0.8, 2.7, 2.8, 3.4, 3.6, 3.9, 2.2, 1.9, 2.4, 3.3, 2.2, 1.1,
    1.9, 0.6, 2.2, 1.9, 2.5, 1.7, 1.9, 1.1, 1.9, 3.1, 1.7, 1.0, 2.9, 2.8, 4.7,
    3.1, 2.4, 1.9, 2.8, 3.1, 2.8, 4.7, 3.1, 3.1, 4.6, 4.4, 1.2, 3.6, 5.8, 6.4,
    4.1, 2.2, 2.9, 1.7, 1.5, 1.5, 1.1, 3.1, 3.6, 3.9, 3.6, 3.4, 2.9, 3.2, 0.5,
    1.0, 2.5, 3.6, 5.5, 5.0, 3.9, 1.1, 2.2, 2.2, 4.1, 1.9, 3.6, 5.8, 6.2, 3.2,
    2.5, 1.4, 0.1, 2.2, 3.6, 4.7, 5.8, 4.4, 3.6, 4.1, 4.4, 3.2, 2.4, 1.7, 1.7,
    2.9, 3.2, 0.7, 1.0, 1.9, 3.1, 4.4, 3.3, 1.4, 0.1, 2.5, 2.9, 0.7, 2.2, 7.5,
    6.9, 4.7, 1.7, 4.1, 1.4, 1.5, 2.4, 1.7, 2.4, 2.8, 3.3, 2.7, 3.6, 2.2, 2.8,
    2.5, 2.4, 2.4, 3.3, 1.2, 3.0, 3.2, 3.6, 2.2, 2.2, 2.9, 3.6, 1.9, 1.2, 2.4,
    1.4, 1.5, 1.2, 2.2, 1.2, 3.3, 2.2, 2.8, 3.3, 1.0, 1.2, 2.7, 2.2])
```

In [7]:
```
KS = sps.kstest(wind_speed, sps.weibull_min(c=2.10701, scale=3.08161).cdf)
KS[1]
```

Out[7]: 0.27512181350825449

Вывод: полученное p-value >= 0.05. Следовательно, гипотезу не отвергаем. То есть наша выборка значений скорости ветра из задания 2 действительно согласуется с распределением Вейбулла.

In [32]:
```
N = 300
K = 200
sample = sps.norm.rvs(size = (K, N), loc=0, scale=1)
X = sample.sum(axis = 1) / N
T = N ** (0.5) * (X)
KS = sps.kstest(T, sps.norm(loc=0, scale=1).cdf)
KS[1]
```

Out[32]: 0.50257320050709686

Вывод: полученное p-value >= 0.05. Следовательно, гипотезу не отвергаем. То есть при больших n распределение статистики из задачи 3.а) задания 2 действительно хорошо приближает предельное распределение.

In [22]:
```
N = 300
K = 200
mu = 1
sample = sps.poisson.rvs(size = (K, N), mu=mu)
X = sample.sum(axis = 1) / N
T = N ** (0.5) * (X - mu)
KS = sps.kstest(T, sps.norm(loc=0, scale=1).cdf)
KS[1]
```

Out[22]: 0.55077431121229425

Вывод: полученное p-value >= 0.05. Следовательно, гипотезу не отвергаем. То есть при больших n распределение статистики из задачи 3.б) задания 2 действительно хорошо приближает предельное распределение.

In [36]:
```
N = 300
K = 200
theta = 1
sample = sps.uniform.rvs(size = (K, N), loc=0, scale=theta)
X = sample.max(axis = 1)
T = N * (theta - X)
KS = sps.kstest(T, sps.expon(scale=theta).cdf)
KS[1]
```

Out[36]: 0.71193648370794116

Вывод: полученное p-value >= 0.05. Следовательно, гипотезу не отвергаем. То есть при больших n распределение статистики из задачи 4 задания 2 действительно хорошо приближает предельное распределение.

In [63]:
```
def check_for_normal_residuals(residuals):
    mean = residuals.mean()
    std = residuals.std()
    KS = sps.kstest(residuals, sps.norm(loc=mean, scale=std).cdf)
    return KS[1]
```

In [96]:
```
print('1) p-value =', check_for_normal_residuals(residuals_1))
print('2) p-value =', check_for_normal_residuals(residuals_2))
print('3) p-value =', check_for_normal_residuals(residuals_3))
print('4) p-value =', check_for_normal_residuals(residuals_4))
print('5) p-value =', check_for_normal_residuals(residuals_5))
print('6) p-value =', check_for_normal_residuals(residuals_6))
print('7) p-value =', check_for_normal_residuals(residuals_7))
```

```
1) p-value = 0.924080529327
2) p-value = 0.932904080761
3) p-value = 0.952334973802
4) p-value = 0.817671576597
5) p-value = 0.992089610709
6) p-value = 0.97438165701
7) p-value = 0.847535312725
```

Вывод: все полученные значения p-value >= 0.05. Следовательно, гипотезу не отвергаем. То есть остатки в регрессии из задач выше нормальны.

In [98]:
```
fr_sample = np.array([97,74,222,20,230,207,45,105,289,391,151,284,363,340,287,442,404,530,470,215,264,171,545,264,171,545,644,86,327,271,86,327,271,86,269,194,258,3
fr_sample
```

Out[98]:
```
array([ 97,  74, 222,  20, 230, 207,  45, 105, 289, 391, 151, 284, 363,
       340, 287, 442, 404, 530, 470, 215, 264, 171, 545, 264, 171, 545,
       644,  86, 327, 271,  86, 327, 271,  86, 269, 194, 258, 363, 119,
        49, 115, 560, 112, 194, 102, 313, 304, 270, 130, 124, 139, 116,
       535,  19])
```

Предположим, что у нашей выборки количества друзей из задания 1 экспоненциальный класс распределений. Пусть это будет наша гипотеза, которую будем сейчас проверять.

In [100]:
```
mean = fr_sample.mean()
minimum = fr_sample.min()
KS = sps.kstest(fr_sample, sps.expon(loc = minimum, scale=mean).cdf)
KS[1]
```

Out[100]:
```
0.2035426280208632
```

Вывод: полученное значение p-value >= 0.05. Следовательно, гипотезу не отвергаем. То есть наша выборка количества друзей из задания 1 имеет экспоненциальный класс распределений.

Задача 8*. Проведите исследование согласно примеру 2 параграфа 2 главы 18 книги М.Б. Лагутина "Наглядная математическая статистика".

In [ ]:
```

```

Задача 9*. Изучите Q-Q plot и критерий Шапиро-Уилка для проверки нормальности, напишите их теоретическое пояснение. В изучении могут помочь материалы курса ПСАД (http://wiki.cs.hse.ru/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%8

Постройте графики Q-Q plot для различных распределений и дайте к ним пояснение. Проверьте различные данные на нормальность с помощью различных критериев и Q-Q plot. Данные можно использовать из задачи 7 или какие-либо еще, например, отдельные компоненты из Ирисов Фишера. Постарайтесь так же правильно контролировать вероятность общей ошибки первого рода.