Случайные процессы. Прикладной поток.

Практическое задание 1

Правила:

- Выполненную работу нужно отправить на почту `probability.diht@yandex.ru`, указав тему письма `"[СП17] Фамилия Имя - Задание 1"`. Квадратные скобки обязательны. Вместо `Фамилия Имя` нужно подставить свои фамилию и имя.
- Прислать нужно ноутбук и его pdf-версию. Названия файлов должны быть такими: `1.N.ipynb` и `1.N.pdf`, где `N` - ваш номер из таблицы с оценками.
- Никакой код из данного задания при проверке запускаться не будет.
- Дедлайн и система оценивания будут объявлены позже.

В Британской империи в Викторианскую эпоху (1837—1901) было обращено внимание на вымирание аристократических фамилий. В связи с этим в своей статье в The Educational Times в 1873 году Гальтон поставил вопрос о вероятности вымирания фамилии. Решение этого вопроса нашел Ватсон и вместе в 1874 году они написали статью "On the probability of the extinction of families". На сайте wikitree.com (http://wikitree.com) в свободно распространяемом формате собрано большое количество данных о родословных различных людей. В коллекции есть как люди, жившие во времена поздней античности, так и наши современники. На основе некоторой части этих данных вам предстоит провести исследование о вымирании фамилий.

Вам предоставляются несколько файлов, в которых содержатся данные о некоторых родословных. Вам предстоит проводить исследование на нескольких из этих файлов (каких именно, см. в таблице). Формат файлов следующий:

```
generation \t name \t gender \t birthday \t deathdate \t parents \t siblings \t
spouses \t children
```

Эти данные означают номер поколения, фамилию, пол, дату рождения, дату смерти, родителей, братьев и сестер, супруг, детей соответственно. Если какая-то характеристика неизвестна (кроме номера поколения и фамилии), вместо нее ставится пустая подстрока. Если каких-то характеристик несколько, то они разделены через ";". Все люди представлены некоторым идентификатором `<id>`, который соответствует адресу `http://www.wikitree.com/wiki/<id>`. Например, идентификатор `Romanov-29` соответствует адресу `http://www.wikitree.com/wiki/Romanov-29` (http://www.wikitree.com/wiki/Romanov-29). В файле родословные отделяются друг от друга пустой строкой.

Для облегчения вашей работы мы предоставляем вам код, который считывает данные из этого файла и преобразует их в список ветвящихся процессов. Каждый ветвящийся процесс содержит список списков, в каждом из которых содержатся все люди из соответствующего поколения. Обратите внимание, что одни и те же родословные могут попасть в разные файлы. В таком случае их можно считать разными, но при желании вы можете удалить копии.

В предоставленных данных в каждой родословной для каждого мужчины на следующем поколении содержатся все его дети, которые были указаны на сайте. Для женщин дети в данной родословной не указаны. Это связано с тем, что женщины обычно меняют свою фамилию, когда выходят замуж, тем самым, они переходят в другую ветку. С точки зрения ветвящихся процессов, нужно иметь в виду, что если у мужчины родилось 3 мальчика и 4 девочки, то у него 3 потомка как продолжателя фамилии.

Ваша задача --- исследовать процесс вымирания фамилий на основе предложенных данных. В данном задании вам предстоит сделать оценку закона размножения, а в следующем задании --- провести остальной анализ.

In [13]:

```
import numpy as np
import scipy.stats as sps
from collections import Counter  # это может пригодиться
from BranchingProcess import Person, BranchingProcess, read_from_files

from statsmodels.sandbox.stats.multicomp import multipletests
from math import factorial, exp

import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams.update({'font.size': 16})
%matplotlib inline
```

1. Описательный анализ

Большая часть кода, необходимая для проведения данного анализа, является технической и основывается на работе с пакетом `BranchingProcess`. Поэтому данный код полностью вам выдается, вам нужно только выполнить его, подставить имена файлов. Кроме того, код анализа позволит вам лучше понять структуру данных.

Считайте данные с помощью предложенного кода. Посчитайте количество родословных.

In [14]:

```
processes = read_from_files(["data/D.txt", "data/W.txt", "data/S.txt",
                             "data/I.txt", "data/N.txt", "data/O.txt",
                             "data/K.txt", "data/J.txt", "data/M.txt",
                             "data/G.txt"])
print(len(processes))
```

```
76628
```

В имеющихся данных очень много людей, про которых известно лишь то, что они когда-то существовали. Обычно их фамилия неизвестна (вместо фамилии у них может стоять, к примеру, `B-290`), а у некоторых из них неизвестен даже пол, не говоря уже о родителях и детях. Такие данные стоит удалить.

Удалите все процессы, состоящие только из одного поколения (в котором, естественно, будет только один человек). Сколько осталось процессов?

In [15]:

```
for i in range(len(processes))[::-1]:
    if len(processes[i].generations) < 2:
        del processes[i]

print(len(processes))
```

```
22166
```

Для лучшего понимания задачи и предложенных данных посчитайте следующие характеристики: минимальное, максимальное и среднее число поколений в роду, год рождения самого старого и самого молодого человека, среднюю продолжительность жизни.

In [16]:

```
generation_counts = []
years = []

for pedigree in processes:
    generation_counts.append(len(pedigree.generations))

    for generation in pedigree.generations:
        for person in generation:
            if person.birthday != '':
                years.append(person.birthday.split('-')[0])

years = np.array(years, dtype=int)
print('Минимальное число поколений в роду:', min(generation_counts))
print('Максимальное число поколений в роду:', max(generation_counts))
print('Среднее число поколений в роду:', round(np.mean(generation_counts), 1))
print('Год рождения самого старого:', min(years))
print('Год рождения самого молодого:', max(years))
```

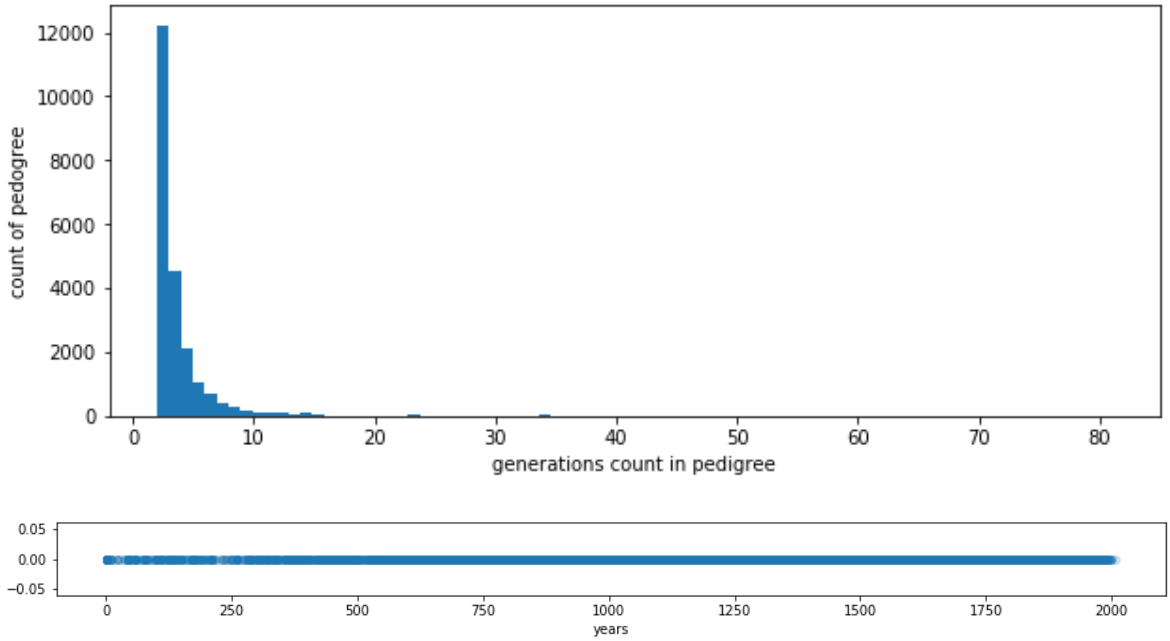
```
Минимальное число поколений в роду: 2
Максимальное число поколений в роду: 81
Среднее число поколений в роду: 3.3
Год рождения самого старого: 1
Год рождения самого молодого: 2007
```

Постройте гистограмму зависимости количества поколений в родословной от количества родословных. На следующем графике отложите на временной оси года рождения всех людей.

In [17]:

```
plt.figure(figsize=(10, 4))
plt.hist(generation_counts, bins=80)
plt.xlabel('generations count in pedigree')
plt.ylabel('count of pedogree')
plt.show()

plt.figure(figsize=(15, 1))
plt.scatter(years, np.zeros_like(years), alpha=0.2)
plt.xlabel('years')
plt.show()
```



Посчитайте среднюю продолжительность жизни.

In [18]:

```
ages = []
for pedigree in processes:
    for generation in pedigree.generations:
        for person in generation:
            if person.birthday != '' and person.deathdate != '':
                ages.append(int(person.deathdate.split('-')[0]) - \
                            int(person.birthday.split('-')[0]))

mean_age = np.mean(ages)
print(round(mean_age, 2))
```

```
56.74
```

2. Оценка закона размножения

Для начала предположим, что все выданные вам процессы являются частью одного большого процесса с общим предком. В следующем задании рассмотрим так же случай, когда все процессы являются разными.

Чтобы проводить какой-либо анализ ветвящегося процесса нужно некоторым образом оценить закон размножения. Кажется, что для этого достаточно посчитать количество сыновей у каждого человека, получив тем самым выборку неотрицательных целых чисел. Однако, проблема в том, что данные неполные, в частности, некоторые поля могут быть не заполнены. Тем не менее обычно у человека указаны либо все дети, либо не указаны вообще. Таким образом, условно мы можем разделить выборку на две части: поле детей заполнено (в т.ч. если у человека на самом деле нет детей), поле детей незаполнено. Если бы первая часть выборки была бы полностью известна, что распределение можно оценить по ней. Нам же неизвестен размер выборки и количество нулевых элементов в ней. Количество положительных элементов известно.

Математическая постановка задачи

P_θ --- неизвестное распределение из некоторого класса распределений $\mathcal{P}$ на $\mathbb{Z}_+$.

$X_1, \ldots, X_n$ --- выборка из распределения P_θ, причем n и количество нулей в выборке неизвестны.

$Y_1, \ldots, Y_s$ --- положительная подвыборка, которая полностью нам известна. В нашей задаче Y_j --- количество сыновей у j-го человека среди тех, у кого есть хотя бы один сын.

Оценку параметра θ можно найти методом максимального правдоподобия:

$$\prod_{i=1}^{s} \mathsf{P}_\theta(Y_i \,|Y_i > 0) \to \max_\theta$$

В качестве классов распределений $\mathcal{P}$ рассмотрите пуассоновское и геометрическое распределения. По желанию можете рассмотреть другие классы распределений, осмысленные в данной задаче

Внимание! Применение метода `fit` из `scipy.stats` является некорректным в данной задаче, поскольку рассматривается усеченная выборка. Задачу максимизации нужно решить явно, выписав все формулы (которые тоже нужно прислать вместе с кодом).

После оценки параметров проведите проверку принадлежности неизвестного распределения рассматриваемому семейству распределений $\mathcal{P}$ с помощью критерия хи-квадрат, взяв для для него то распределение из $\mathcal{P}$, которое соответствует оценке максимального правдоподобия. Постарайтесь учесть все особенности проверки гипотез, которые обсуждались на семинаре. Для каждого класса постройте также график частот и функции $\mathsf{P}_\theta(y \,|Y > 0)$.

1. Пуассоновское распределение

Функция вероятности Пуассоновского распределения есть

$$p_\theta(k) = \frac{\theta^k}{k!} exp^{-\theta}$$

Так как рассматриваются только положительные значения, то будем работать с функцией

$$P_\theta(k \mid k > 0) = \frac{\theta^k}{k!(exp^\theta - 1)}$$

Тогда необходимо максимизировать функцию

$$f = \frac{\theta^{\sum_{i=1}^{s} Y_i}}{\prod_{i=1}^{s} Y_i!} \frac{1}{(exp^\theta - 1)^s}$$

Прологарифмировав и взяв производную, получаем

$$L' = \frac{\sum_{i=1}^{s} Y_i}{\theta} - s\frac{exp^{\theta}}{exp^{\theta} - 1} = 0$$

Функция

$$f(\theta) = \frac{\theta}{1 - exp^{-\theta}} - \overline{Y}$$

является монотонной. Следовательно, θ можно найти с помощью бинарного поиска.

2. Геометрическое распределение

Функция вероятности геометрического распределения есть

$$p_\theta(k) = p(1-p)^k,$$

где k - номер первого успеха.

Так как рассматриваются только положительные значения, то будем работать с функцией

$$p_\theta(k \mid k > 0) = p(1-p)^{k-1}$$

Тогда необходимо максимизировать функцию

$$f = \frac{p^s(1-p)^{\sum_{i=1}^{s} Y_i}}{(1-p)^s}$$

Прологарифмировав и взяв производную, получаем

$$L = \frac{s}{p} - \frac{-s + \sum_{i=1}^{s} Y_i}{1-p}$$

Таким образом, оценкой параметра θ является $\theta^* = \frac{1}{\overline{Y}}$.

In [19]:

```
# Генерация выборки
# Смотрим на мужчин в поколении i и проверяем их детей в поколении i + 1

sample = []

for pedigree in processes:
    for i in range(len(pedigree.generations) - 1):
        for person in pedigree.generations[i]:
            if person.gender == "male" and (len(person.children) > 0):
                tmp = 0
                for child in pedigree.generations[i+1]:
                    if child.gender == "male" and \
                    (child.name in person.children):
                        tmp += 1
                if (tmp != 0):
                    sample.append(tmp)

sample = np.array(sample)
theta_geom = 1 / np.mean(sample)
```

In [20]:

```
print('Оценка параметра для геометрического распределения: ', round(theta_geom, 2))
```

```
Оценка параметра для геометрического распределения:  0.46
```

In [21]:

```
def check_geom(x, theta, bins=10, alpha=0.05):
    # print("Гипотеза: распределение геометрическое.\n")

    # p(1 - p) ^ (k - 1) - функция вероятности

    n = len(x)
    i = 1
    bad = 0
    res = []

    while True:
        tmp = n * theta * pow(1 - theta, i - 1)
        if tmp > 5:
            res.append(tmp)
            bad += theta * pow(1 - theta, i - 1)
            i += 1
        else:
            res.append(n * (1 - bad))
            break

    test = []

    for i in range(1, len(res)):
        tmp = 0
        for j in x:
            if j == i:
                tmp += 1
        test.append(tmp)

    test.append(n - sum(test))

    ctest = sps.chisquare(test, res)

    return ctest.pvalue

    # Строки закомментированы, чтобы функцию удобно
    # было использовать для multipletests
    # if ctest.pvalue < alpha:
    #     print ("Гипотеза отвергается.\n")
    # else:
    #     print ("Гипотеза не отвергается.\n")
```

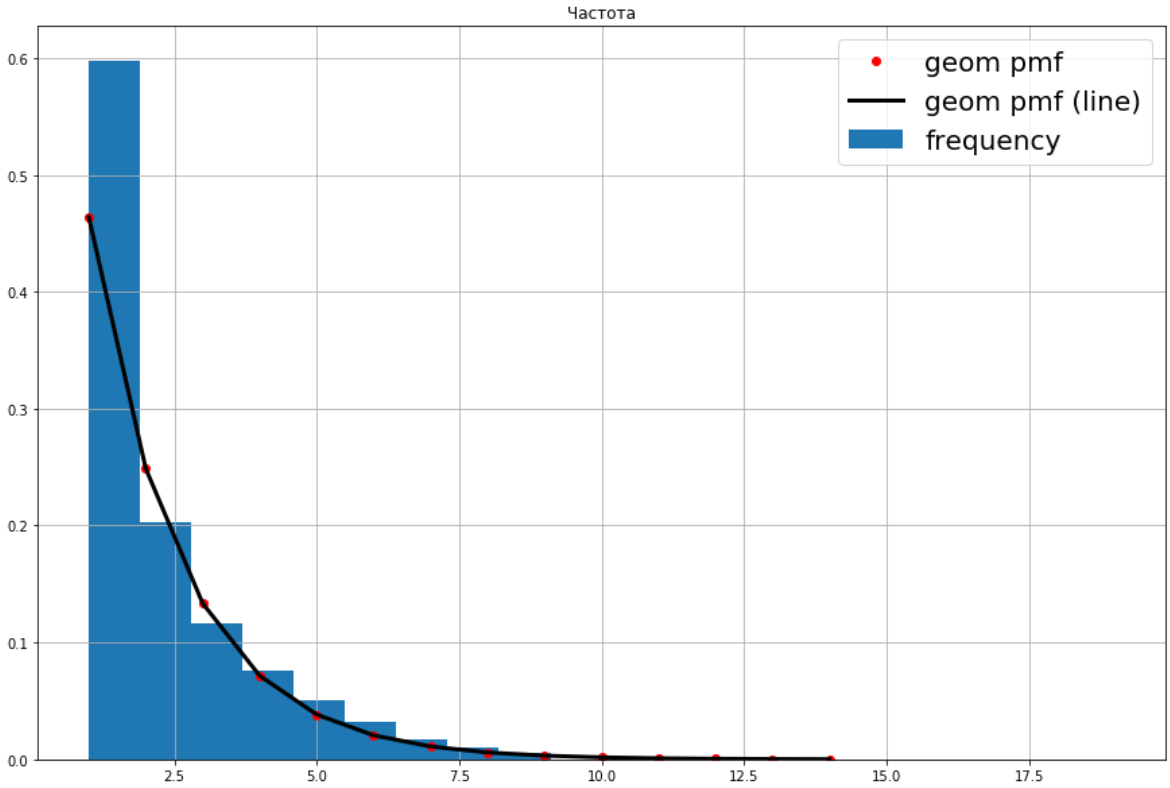
In [35]:

```
x = np.arange(1, 15, 1)
y = list(map(lambda t: theta_geom * pow(1 - theta_geom, t - 1), x))

plt.figure(figsize=(15,10))

plt.hist(sample, bins=20, normed=True, label='frequency')
plt.plot(x, y, 'ro', label='geom pmf', linewidth=3)
plt.plot(x, y, c='black', label='geom pmf (line)', linewidth=3)

plt.legend(prop={'size': 20})
plt.title('Частота')
plt.grid()
plt.show()
```



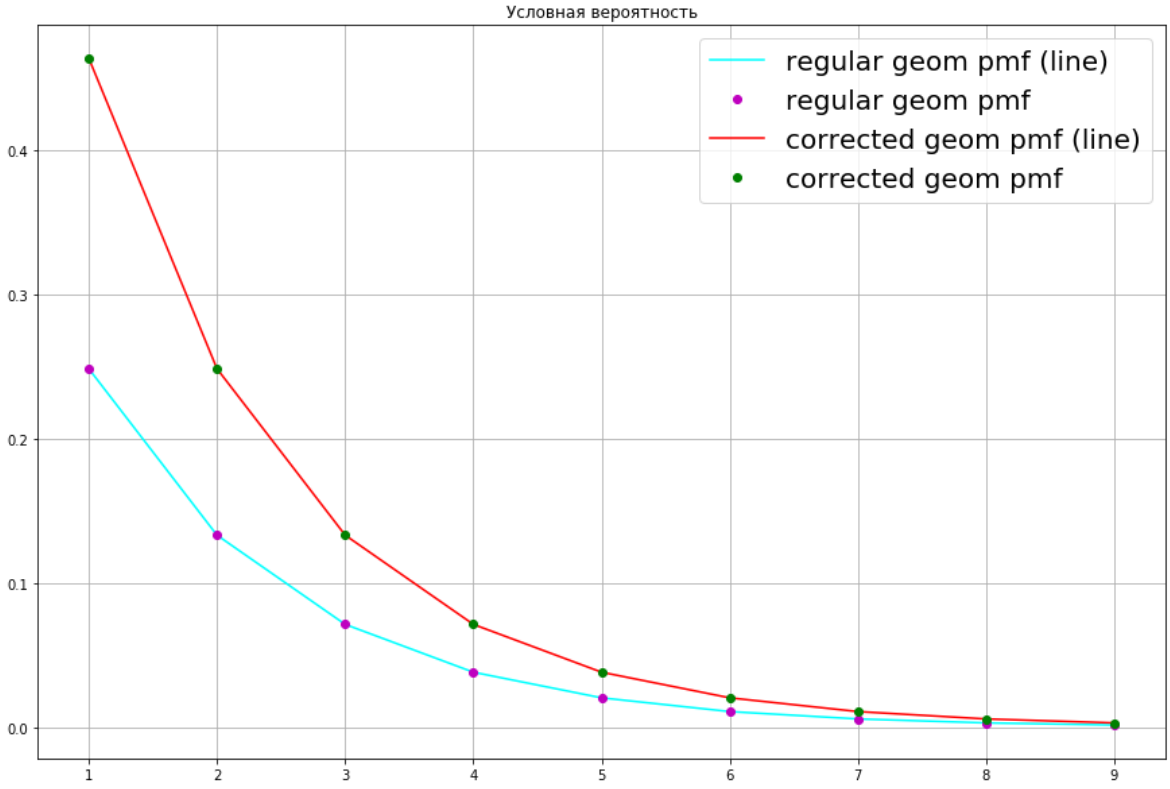
In [23]:

```
x = np.arange(1, 10)
y = list(map(lambda t: theta_geom * pow(1 - theta_geom, t - 1), x))
yy = list(map(lambda t: theta_geom * pow(1 - theta_geom, t), x))

plt.figure(figsize=(15,10))

plt.plot(x, yy, c='cyan', label='regular geom pmf (line)')
plt.plot(x, yy, 'mo', label='regular geom pmf')
plt.plot(x, y, c='red', label='corrected geom pmf (line)')
plt.plot(x, y, 'go', label='corrected geom pmf')

plt.legend(prop={'size': 20})
plt.title('Условная вероятность')
plt.grid()
plt.show()
```



In [24]:

```
extra = np.mean(sample)

def bin_search(left, right):
    if (right - left < 1e-5):
        return (right + left) / 2
    elif (right - left > 1e-5):
        mn = (right + left) / 2
        mn_value = mn / (1 - exp(-mn)) - extra
        if mn_value < 0:
            return bin_search(mn, right)
        else:
            return bin_search(left, mn)
```

In [25]:

```
theta_pois = bin_search(0, 100)
print('Оценка параметра для Пуассоновского распределения: ',
      round(theta_pois, 2))
```

```
Оценка параметра для Пуассоновского распределения:  1.8
```

In [26]:

```
def check_pois(x, theta, bins=10, alpha=0.05):
    # print("Гипотеза: распределение Пуассона.\n")

    # theta ^ k / (k! * (e ^ theta - 1)) - функция вероятности

    n = len(x)
    i = 1
    bad = 0
    res = []

    while True:
        tmp = n * pow(theta, i) / ((exp(theta) - 1) * factorial(i))
        if tmp > 5:
            res.append(tmp)
            bad += pow(theta, i) / ((exp(theta) - 1) * factorial(i))
            i += 1
        else:
            res.append(n * (1 - bad))
            break

    test = []

    for i in range(1, len(res)):
        tmp = 0
        for j in x:
            if j == i:
                tmp += 1
        test.append(tmp)

    test.append(n - sum(test))

    #print(test)
    #print(res)

    ctest = sps.chisquare(test, res)

    #print(ctest.pvalue)
    return ctest.pvalue

    # Строки закомментированы, чтобы функцию удобно
    # было использовать для multipletests
    # if ctest.pvalue < alpha:
    #     print ("Гипотеза отвергается.\n")
    # else:
    #     print ("Гипотеза не отвергается.\n")
```

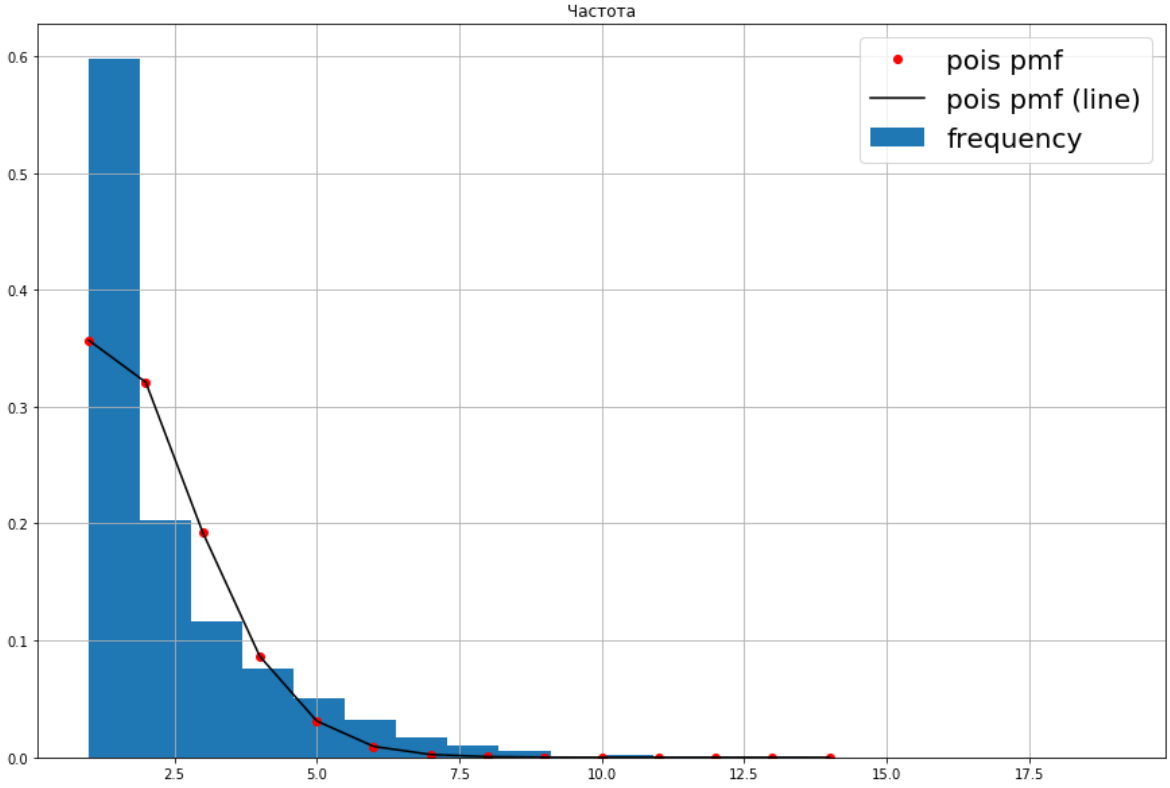
In [36]:

```
x = range(1, 15)
y = list(map(lambda t: pow(theta_pois, t) / \
             (factorial(t) * (exp(theta_pois) - 1)), x))

plt.figure(figsize=(15,10))

plt.hist(sample, bins=20, normed=True, label='frequency')
plt.plot(x, y, 'ro', label='pois pmf')
plt.plot(x, y, c='black', label='pois pmf (line)')

plt.legend(prop={'size': 20})
plt.title('Частота')
plt.grid()
plt.show()
```



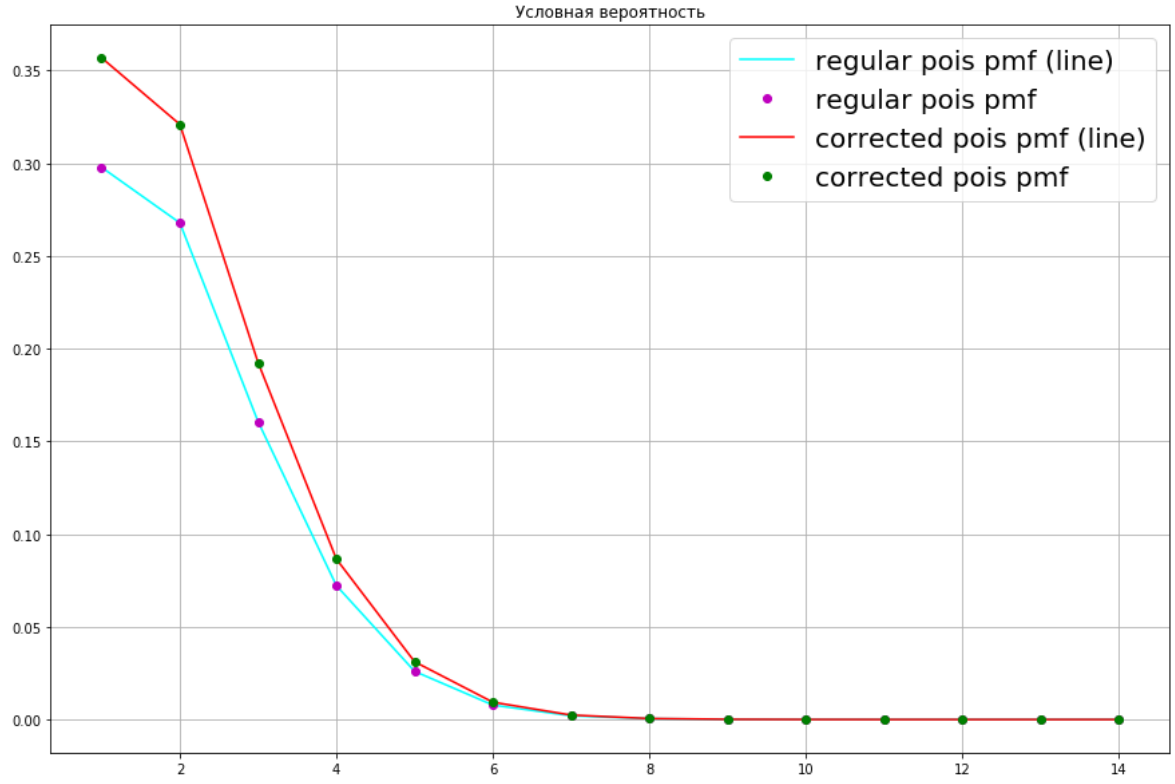
In [37]:

```
x = range(1, 15)
y = list(map(lambda t: pow(theta_pois, t) / \
             (factorial(t) * (exp(theta_pois) - 1)), x))
yy = sps.poisson.pmf(x, theta_pois)

plt.figure(figsize=(15,10))

plt.plot(x, yy, c='cyan', label='regular pois pmf (line)')
plt.plot(x, yy, 'mo', label='regular pois pmf')
plt.plot(x, y, c='red', label='corrected pois pmf (line)')
plt.plot(x, y, 'go', label='corrected pois pmf')

plt.legend(prop={'size': 20})
plt.title('Условная вероятность')
plt.grid()
plt.show()
```



In [34]:

```
pvs_g = []
pvs_p = []

for i in range(10):
    sub_x = sample[sps.randint.rvs(0, len(sample), size=150)]
    pvs_g.append(check_geom(sub_x, theta_geom))

pvs_g = np.array(pvs_g)

for i in range(10):
    sub_x = sample[sps.randint.rvs(0, len(sample), size=150)]
    pvs_p.append(check_pois(sub_x, theta_pois))

pvs_p = np.array(pvs_p)

pvs = np.append(pvs_g, pvs_p)

res = multipletests(pvs, method='bonferroni')
print("Гипотеза отклоняется:", res[0])
print("Скорректированные p-values:",res[1])
```

```
Гипотеза отклоняется: [False False False False False False False False
False False  True  True
  True  True  True  True  True  True  True  True]
Скорректированные p-values: [  1.00000000e+00   1.00000000e+00   1.000
00000e+00   1.00000000e+00
   1.00000000e+00   1.68464138e-01   2.05929493e-01   1.00000000e+00
   2.27718662e-01   8.54456317e-01   7.68157530e-08   8.22027210e-09
   8.74750076e-05   2.38412084e-03   1.05218917e-08   2.38671266e-10
   3.26783563e-10   1.71385337e-11   3.49229543e-06   2.28874764e-09]
```

Вывод: Множественная проверка гипотез отвергает Пуассоновское распределение и не отвергает геометрическое.

In [ ]: