Критерий Хи-квадрат для проверки нормальности

In [180]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [181]:
```
with open("C:\\Users\\admin\\Dropbox\\5 семестр\\ТВ и Мат Стат\\11.7.txt") as file:
    a = np.array(list(map(float, file.read().replace(',', '.').split())))
```

In [182]:
```
len(a)
```

Out[182]:
```
100
```

In [183]:
```
sum(a)
```

Out[183]:
```
1194.4200000000003
```

In [184]:
```
b = np.sort(a)
b
```

Out[184]:
```
array([10.4 , 10.53, 10.8 , 10.85, 10.98, 10.99, 11.03, 11.13, 11.3 ,
       11.32, 11.34, 11.38, 11.38, 11.39, 11.4 , 11.47, 11.5 , 11.55,
       11.56, 11.57, 11.57, 11.6 , 11.62, 11.62, 11.65, 11.68, 11.7 ,
       11.71, 11.72, 11.72, 11.74, 11.75, 11.79, 11.79, 11.81, 11.81,
       11.81, 11.86, 11.87, 11.88, 11.89, 11.89, 11.9 , 11.92, 11.92,
       11.92, 11.93, 11.94, 11.96, 11.97, 11.98, 11.98, 11.98, 11.99,
       12.  , 12.  , 12.05, 12.07, 12.08, 12.1 , 12.1 , 12.1 , 12.1 ,
       12.11, 12.11, 12.11, 12.15, 12.15, 12.15, 12.16, 12.17, 12.18,
       12.19, 12.19, 12.23, 12.36, 12.37, 12.38, 12.39, 12.39, 12.42,
       12.42, 12.42, 12.42, 12.44, 12.45, 12.45, 12.45, 12.45, 12.46,
       12.58, 12.6 , 12.67, 12.7 , 12.77, 12.86, 12.88, 12.93, 12.95,
       12.97])
```

Равночастотные интервалы

In [185]:
```
# ni - кол-во элементов выборки в интервале
# n - кол-во интервалов
ni = 10
n = int(len(a) / ni)
n
```

Out[185]:
```
10
```

In [186]:
```
import math
from scipy import stats
```

Интервалы вида (;](;]...(;](;)

In [187]:
```python
# intervals - интервалы разбиения (n штук)
intervals = [(float('-inf'), b[9])]

for i in range(1, n - 1):
    intervals.append((b[i * ni - 1], b[i * ni + (ni - 1)]))

intervals.append((b[len(b) - (ni + 1)], float('+inf')))
intervals
```

Out[187]:
```
[(-inf, 11.32),
 (11.32, 11.57),
 (11.57, 11.72),
 (11.72, 11.88),
 (11.88, 11.97),
 (11.97, 12.1),
 (12.1, 12.16),
 (12.16, 12.39),
 (12.39, 12.46),
 (12.46, inf)]
```

In [188]:
```python
MX = sum(a) / len(a)
DX = sum((a - MX) ** 2) / len(a)
S = math.sqrt(DX)
normal = stats.norm(loc=MX, scale=S)
```

In [189]:
```python
l2 = []
for i in range(n):
    l2.append(normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]))
l2
```

Out[189]:
```
[0.11053455639017529,
 0.12106574901997796,
 0.09854037259543313,
 0.11978126594237304,
 0.07024742716785409,
 0.09981222376029597,
 0.043891666616652,
 0.1450546542490737,
 0.03510726931060486,
 0.1559648149475643]
```

In [190]:
```python
sum(l2)
```

Out[190]:
```
1.0
```

$$\left(\sum_{i=1}^{n} n_i ln(p_i(\theta))\right) \to \max_{\theta}$$

$$\forall i \ n_i = 10$$

$$\Rightarrow \sim \left(\sum_{i=1}^{n} ln(p_i(\theta))\right) \to \max_{\theta}$$

In [191]:
```python
from scipy.optimize import Bounds
from scipy.optimize import minimize
```

In [192]:
```python
# начальный
intervals[0][0], intervals[0][1]
```

Out[192]:
```
(-inf, 11.32)
```

In [193]:
```python
MX, S
```

Out[193]:
```
(11.944200000000002, 0.5100964222575962)
```

In [194]:

```python
def lnL(parameters):
    mu = parameters[0]
    sigma = parameters[1]
  #  print(mu, sigma)
    normal = stats.norm(loc=mu, scale=sigma)
    l = [normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]) for i in range(n)]
  #  print(l)
    s = sum(np.log(l))
  #  print(s)
    return -s

MX_0 = MX
S_0 = S

# пределы мат ожидания от минимального до максимального по выборке
# для дисперсии от 0 до inf
bounds = Bounds([b[0], 0], [b[99], +float('inf')])

for method in ['L-BFGS-B', 'TNC', 'SLSQP']:
    print(method, ':')
    res = minimize(lnL, np.asarray([MX_0, S_0]), method=method, tol=1e-6, options={'maxiter': 500}, bounds
    print(res, '\n')

print('Nelder-Mead:')
res = minimize(lnL, np.asarray([MX_0, S_0]), method="Nelder-Mead", tol=1e-6, options={'maxiter': 500})
print(res, '\n')
```

```
L-BFGS-B :

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: divide by zero encou
ntered in log


      fun: nan
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([-0.17064501,  2.94475377])
  message: b'ABNORMAL_TERMINATION_IN_LNSRCH'
     nfev: 63
      nit: 0
   status: 2
  success: False
        x: array([11.9442    ,  0.51009642])

TNC :
     fun: 23.884441648921186
     jac: array([-0.00083382, -0.00035136])
 message: 'Converged (|f_n-f_(n-1)| ~= 0)'
    nfev: 26
     nit: 5
  status: 1
 success: True
       x: array([11.94882472,  0.45284227])

SLSQP :
     fun: 23.884441644874816
     jac: array([-0.00024104,  0.00071192])
 message: 'Optimization terminated successfully.'
    nfev: 23
     nit: 5
    njev: 5
  status: 0
 success: True
       x: array([11.9488374 ,  0.45285777])

Nelder-Mead:
 final_simplex: (array([[11.94884318,  0.45284762],
       [11.94884274,  0.45284813],
       [11.94884224,  0.4528467 ]]), array([23.88444164, 23.88444164, 23.88444164]))
           fun: 23.884441640572298
       message: 'Optimization terminated successfully.'
          nfev: 87
           nit: 45
        status: 0
       success: True
```

```
        x: array([11.94884318,  0.45284762])
```

In [195]:
```python
def lnL2(parameters):
    mu = parameters[0]
    sigma = parameters[1]
  #  print(mu, sigma)
    normal = stats.norm(loc=mu, scale=sigma)
    l = [normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]) for i in range(n)]
  #  print(l)
    return -np.prod(l)


MX_0 = MX
S_0 = S

# пределы мат ожидания от минимального до максимального по выборке
# для дисперсии от 0 до inf
bounds = Bounds([b[0], 0], [b[99], +float('inf')])

for method in ['L-BFGS-B', 'TNC', 'SLSQP']:
    print(method, ':')
    res = minimize(lnL2, np.asarray([MX_0, S_0]), method=method, tol=1e-6, options={'maxiter': 500}, bounc
    print(res, '\n')

print('Nelder-Mead:')
res = minimize(lnL2, np.asarray([MX_0, S_0]), method="Nelder-Mead", tol=1e-6, options={'maxiter': 500})
print(res, '\n')
```

```
L-BFGS-B :
      fun: -3.860771014767754e-11
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([-6.58818403e-12,  1.13690204e-10])
  message: b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     nfev: 3
      nit: 0
   status: 0
  success: True
        x: array([11.9442    ,  0.51009642])

TNC :
     fun: -3.860771014767754e-11
     jac: array([-6.58818403e-12,  1.13690204e-10])
 message: 'Local minimum reached (|pg| ~= 0)'
    nfev: 1
     nit: 0
  status: 0
 success: True
       x: array([11.9442    ,  0.51009642])

SLSQP :
     fun: -3.860771014767754e-11
     jac: array([-6.58818963e-12,  1.13690206e-10])
 message: 'Optimization terminated successfully.'
    nfev: 4
     nit: 1
    njev: 1
  status: 0
 success: True
       x: array([11.9442    ,  0.51009642])

Nelder-Mead:
 final_simplex: (array([[11.94884318,  0.45284762],
       [11.94884274,  0.45284813],
       [11.94884224,  0.4528467 ]]), array([-4.23758861e-11, -4.23758861e-11, -4.23758861e-11]))
           fun: -4.2375886055921025e-11
       message: 'Optimization terminated successfully.'
          nfev: 87
           nit: 45
        status: 0
       success: True
             x: array([11.94884318,  0.45284762])
```

In [196]:
```python
def lnL(parameters):
    mu = parameters[0]
    sigma = parameters[1]
  #  print(mu, sigma)
    normal = stats.norm(loc=mu, scale=sigma)
    l = [normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]) for i in range(n)]
  #  print(l)
    s = sum(np.log(l))
  #  print(s)
    return -s

MX_0 = MX
S_0 = S

# пределы мат ожидания от минимального до максимального по выборке
# для дисперсии от 0 до 1e-3
bounds = Bounds([b[0], 0], [b[99], 1e-3])

for method in ['L-BFGS-B', 'TNC', 'SLSQP']:
    print(method, ':')
    res = minimize(lnL, np.asarray([MX_0, S_0]), method=method, tol=1e-6, options={'maxiter': 500}, bounds
    print(res, '\n')

print('Nelder-Mead:')
res = minimize(lnL, np.asarray([MX_0, S_0]), method="Nelder-Mead", tol=1e-6, options={'maxiter': 500})
print(res, '\n')
```

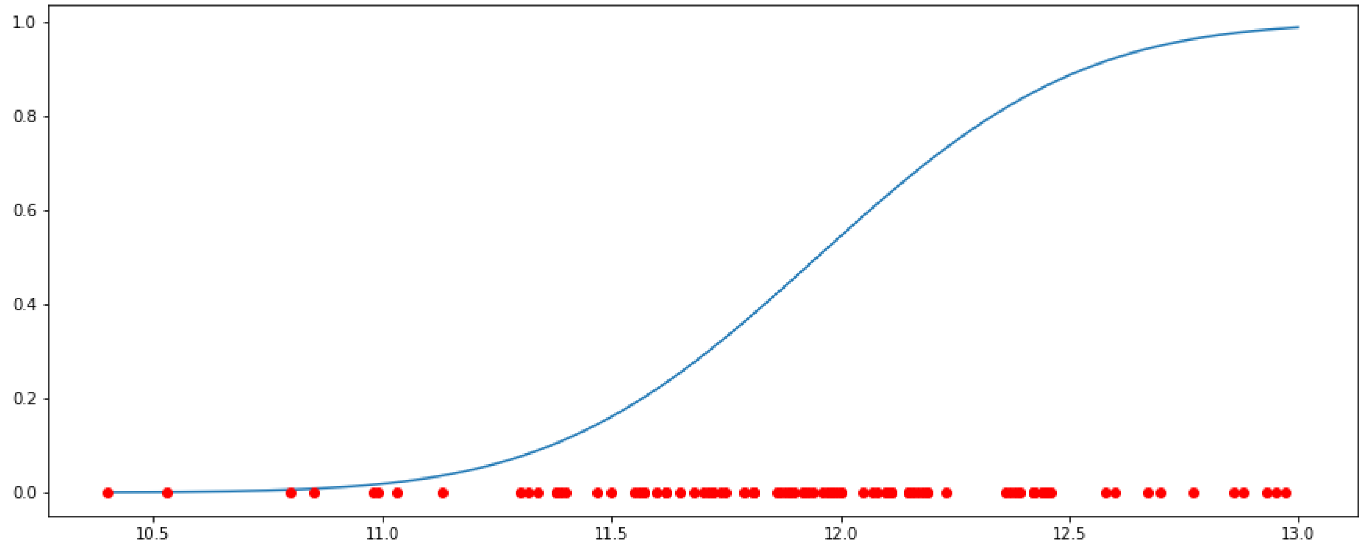
```
L-BFGS-B :

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: divide by zero encou
ntered in log


      fun: nan
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([nan, nan])
  message: b'ABNORMAL_TERMINATION_IN_LNSRCH'
     nfev: 63
      nit: 0
   status: 2
  success: False
        x: array([1.19442e+01, 1.00000e-03])

TNC :
     fun: inf
     jac: array([nan, nan])
 message: 'Unable to progress'
    nfev: 499
     nit: 498
  status: 6
 success: False
       x: array([1.19442e+01, 1.00000e-03])

SLSQP :
     fun: nan
     jac: array([nan, nan])
 message: 'Iteration limit exceeded'
    nfev: 7004
     nit: 501
    njev: 501
  status: 9
 success: False
       x: array([nan, nan])

Nelder-Mead:
 final_simplex: (array([[11.94884318,  0.45284762],
       [11.94884274,  0.45284813],
       [11.94884224,  0.4528467 ]]), array([23.88444164, 23.88444164, 23.88444164]))
           fun: 23.884441640572298
       message: 'Optimization terminated successfully.'
          nfev: 87
           nit: 45
        status: 0
       success: True
             x: array([11.94884318,  0.45284762])
```

In [197]:

```python
def lnL2(parameters):
    mu = parameters[0]
    sigma = parameters[1]
  #  print(mu, sigma)
    normal = stats.norm(loc=mu, scale=sigma)
    l = [normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]) for i in range(n)]
  #  print(l)
    return -np.prod(l)

MX_0 = MX
S_0 = S

# пределы мат ожидания от минимального до максимального по выборке
# для дисперсии от 0 до 1e-3
bounds = Bounds([b[0], 0], [b[99], 1e-3])

for method in ['L-BFGS-B', 'TNC', 'SLSQP']:
    print(method, ':')
    res = minimize(lnL2, np.asarray([MX_0, S_0]), method=method, tol=1e-6, options={'maxiter': 500}, bounc
    print(res, '\n')

print('Nelder-Mead:')
res = minimize(lnL2, np.asarray([MX_0, S_0]), method="Nelder-Mead", tol=1e-6, options={'maxiter': 500})
print(res, '\n')
```

```
L-BFGS-B :
      fun: -0.0
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>
      jac: array([0., 0.])
  message: b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL'
     nfev: 3
      nit: 0
   status: 0
  success: True
        x: array([1.19442e+01, 1.00000e-03])

TNC :
     fun: -0.0
     jac: array([0., 0.])
 message: 'Local minimum reached (|pg| ~= 0)'
    nfev: 1
     nit: 0
  status: 0
 success: True
       x: array([1.19442e+01, 1.00000e-03])

SLSQP :
     fun: -0.0
     jac: array([0., 0.])
 message: 'Optimization terminated successfully.'
    nfev: 4
     nit: 1
    njev: 1
  status: 0
 success: True
       x: array([1.19442e+01, 1.00000e-03])

Nelder-Mead:
 final_simplex: (array([[11.94884318,  0.45284762],
       [11.94884274,  0.45284813],
       [11.94884224,  0.4528467 ]]), array([-4.23758861e-11, -4.23758861e-11, -4.23758861e-11]))
           fun: -4.2375886055921025e-11
       message: 'Optimization terminated successfully.'
          nfev: 87
           nit: 45
        status: 0
       success: True
             x: array([11.94884318,  0.45284762])
```

In [198]:
```
from scipy.stats import norm
x_axis = np.arange(10.4, 13, 0.001)

plt.subplots(figsize=(15, 6))

plt.plot(x_axis, norm.cdf(x_axis, res.x[0], res.x[1]))
plt.plot(b, [0] * len(b), 'ro')
plt.show()
```



In [199]:
```
res.x
```

Out[199]:
```
array([11.94884318,  0.45284762])
```

In [200]:
```
MM, SS = res.x
MM, SS
```

Out[200]:
```
(11.948843181407483, 0.4528476174465836)
```

In [201]:
```
normal = stats.norm(loc=MM, scale=SS)
```

In [202]:
```
l3 = []
for i in range(n):
    l3.append(normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]))
# p i -oe
l3 = np.asarray(l3)
l3
```

Out[202]:
```
array([0.08247083, 0.11894356, 0.10524473, 0.13292534, 0.07904715,
       0.11210005, 0.04876319, 0.15552251, 0.03548335, 0.12949929])
```

In [203]:
```
sum(l3)
```

Out[203]:
```
1.0
```

In [204]:
```
b
```

Out[204]:
```
array([10.4 , 10.53, 10.8 , 10.85, 10.98, 10.99, 11.03, 11.13, 11.3 ,
       11.32, 11.34, 11.38, 11.38, 11.39, 11.4 , 11.47, 11.5 , 11.55,
       11.56, 11.57, 11.57, 11.6 , 11.62, 11.62, 11.65, 11.68, 11.7 ,
       11.71, 11.72, 11.72, 11.74, 11.75, 11.79, 11.79, 11.81, 11.81,
       11.81, 11.86, 11.87, 11.88, 11.89, 11.89, 11.9 , 11.92, 11.92,
       11.92, 11.93, 11.94, 11.96, 11.97, 11.98, 11.98, 11.98, 11.99,
       12.  , 12.  , 12.05, 12.07, 12.08, 12.1 , 12.1 , 12.1 , 12.1 ,
       12.11, 12.11, 12.11, 12.15, 12.15, 12.15, 12.16, 12.17, 12.18,
       12.19, 12.19, 12.23, 12.36, 12.37, 12.38, 12.39, 12.39, 12.42,
       12.42, 12.42, 12.42, 12.44, 12.45, 12.45, 12.45, 12.45, 12.46,
       12.58, 12.6 , 12.67, 12.7 , 12.77, 12.86, 12.88, 12.93, 12.95,
       12.97])
```

In [205]:
```
# ni - количество элементов выборки в интревале
# n - количество интервалов
ni, n, len(b)
```
Out[205]:
```
(10, 10, 100)
```
In [206]:
```
for i in range(n):
    print("(", ni, "-", l3[i] * len(b), ") ** 2 /", l3[i] * len(b))
S_hi2 = (ni - len(b) * l3) ** 2 / len(b) / l3
```
```
( 10 - 8.247083292653723 ) ** 2 / 8.247083292653723
( 10 - 11.894356147111063 ) ** 2 / 11.894356147111063
( 10 - 10.524472761780878 ) ** 2 / 10.524472761780878
( 10 - 13.292533818923275 ) ** 2 / 13.292533818923275
( 10 - 7.904714932391915 ) ** 2 / 7.904714932391915
( 10 - 11.210005239394249 ) ** 2 / 11.210005239394249
( 10 - 4.876318733430429 ) ** 2 / 4.876318733430429
( 10 - 15.552251051658805 ) ** 2 / 15.552251051658805
( 10 - 3.548334733300096 ) ** 2 / 3.548334733300096
( 10 - 12.949929289355566 ) ** 2 / 12.949929289355566
```
In [207]:
```
S_hi2 = sum(S_hi2)
S_hi2
```
Out[207]:
```
21.970305917379303
```
In [208]:
```
from scipy.stats import chi2
p = 0.95 # 1 - alpha, alpha = 0.05
# 2 параметра оценил
df = 21 - 2 - 1
value = chi2.ppf(p, df)
value
```
Out[208]:
```
28.869299430392623
```
In [209]:
```
S_hi2 < value
```
Out[209]:
```
True
```
Гипотеза принимается

In [ ]:

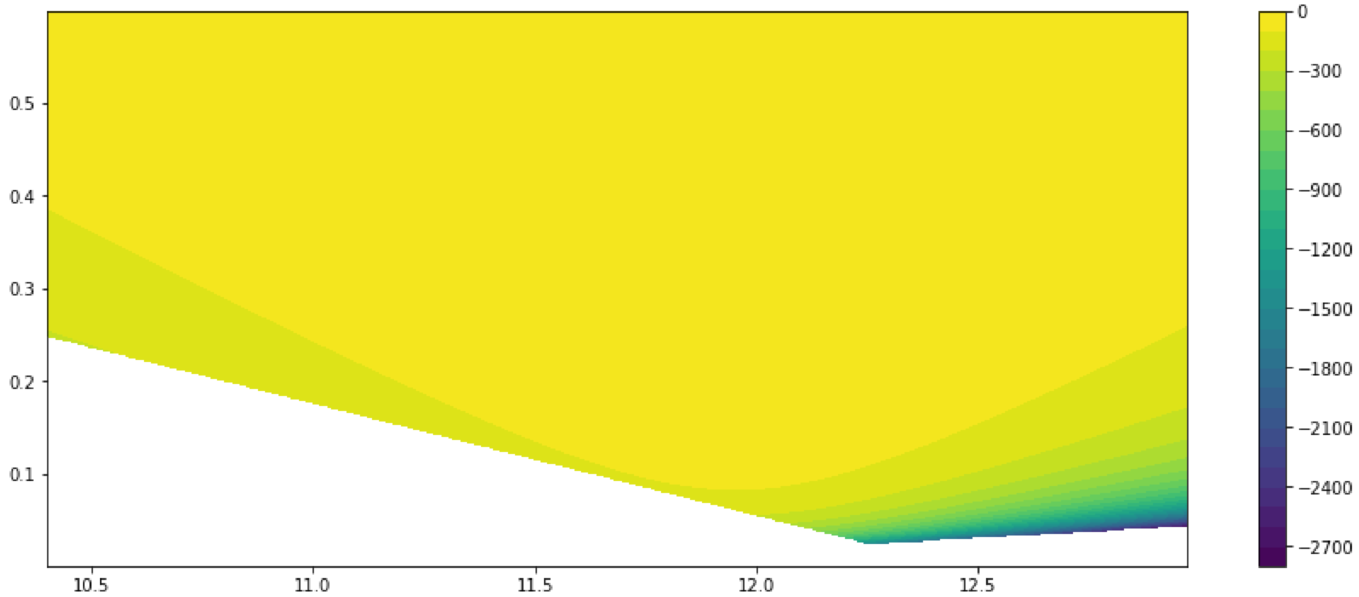
Эмпирически

In [210]:
```
def lnL(parameters):
    mu = parameters[0]
    sigma = parameters[1]
#   print(mu, sigma)
    normal = stats.norm(loc=mu, scale=sigma)
    l = [normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]) for i in range(n)]
#   print(l)
    s = sum(np.log(l))
#   print(s)
    return s
```

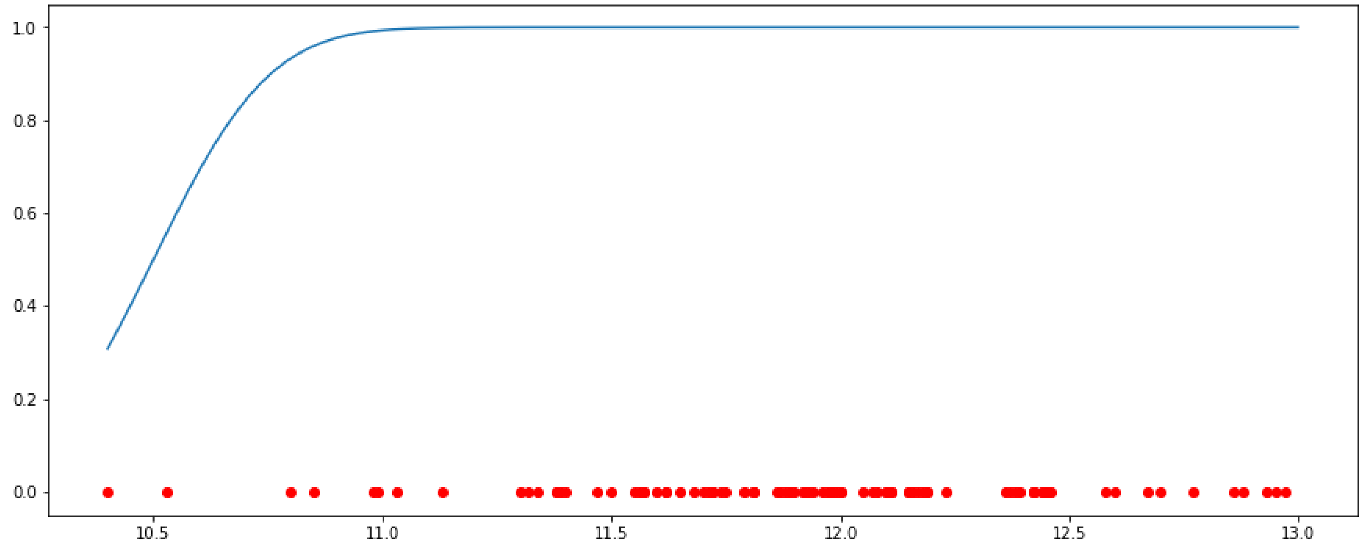
In [211]:
```
plt.subplots(figsize=(15, 6))

x = np.arange(b[0], b[99], 0.001)
y = np.arange(0.001, 0.6, 0.001)
xx, yy = np.meshgrid(x, y, sparse=True)
z = lnL((xx, yy))
cs = plt.contourf(x, y, z, 30, cmap='viridis')
#plt.clabel(cs, colors='k', inline=False)
plt.colorbar()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: divide by zero encou
ntered in log
```

Out[211]: <matplotlib.colorbar.Colorbar at 0x3d6ffad0>



In [212]:
```
from scipy.stats import norm
x_axis = np.arange(10.4, 13, 0.001)

plt.subplots(figsize=(15, 6))

plt.plot(x_axis, norm.cdf(x_axis, 10.5, 0.2))
plt.plot(b, [0] * len(b), 'ro')
plt.show()
```



In [213]:
```
lnL((10.5, 0.2))
```

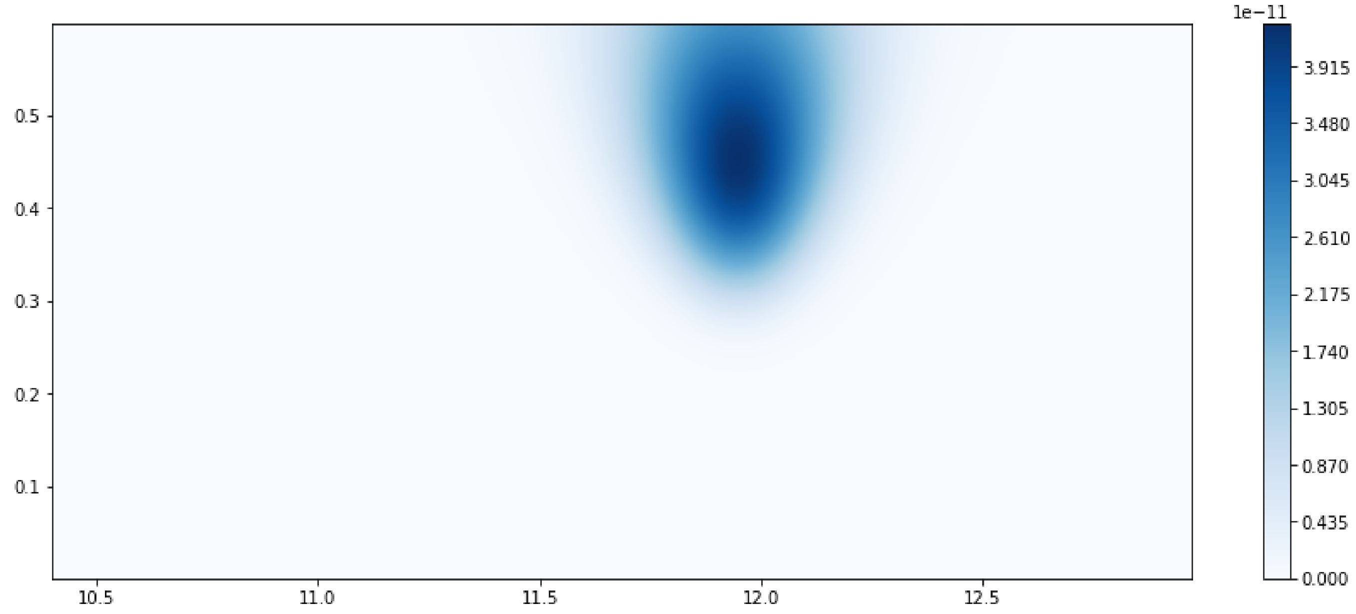
```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: RuntimeWarning: divide by zero encou
ntered in log
```

Out[213]:
```
-inf
```

In [214]:
```
def lnL2(parameters):
    mu = parameters[0]
    sigma = parameters[1]
   # print(mu, sigma)
    normal = stats.norm(loc=mu, scale=sigma)
    l = [normal.cdf(intervals[i][1]) - normal.cdf(intervals[i][0]) for i in range(n)]
  #  print(l)
    k = 1
    for i in range(len(l)):
        k *= l[i]
    return k
```

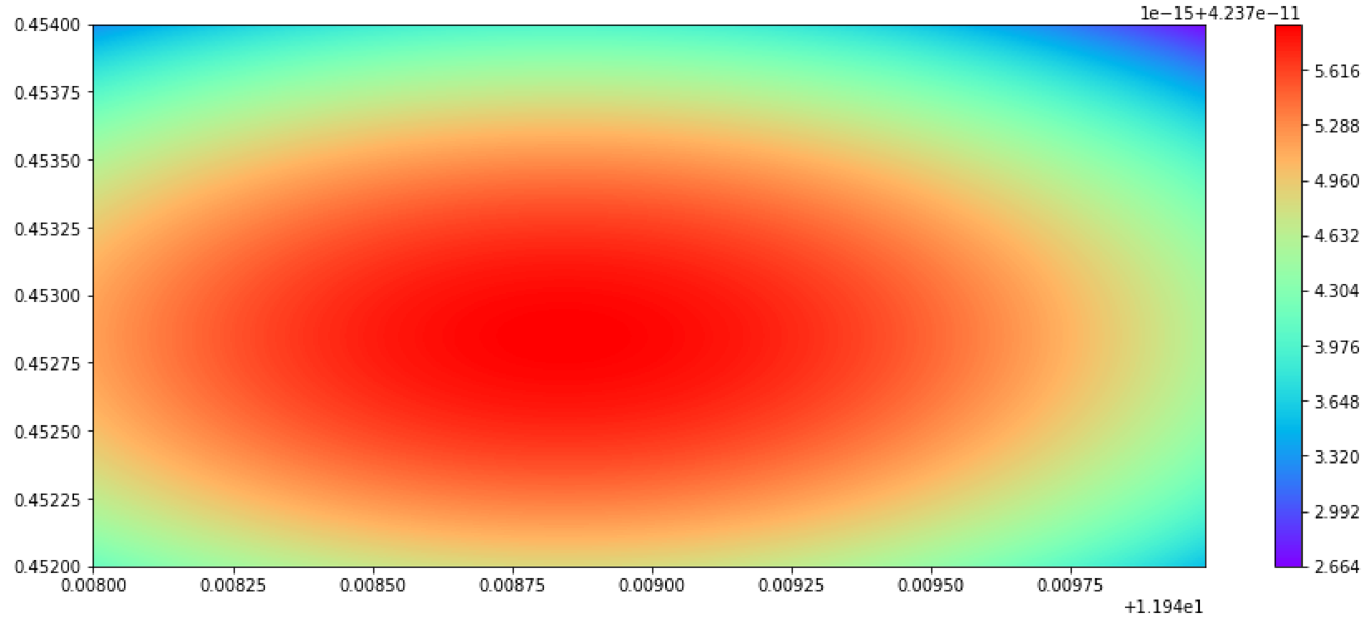
In [218]:
```
plt.subplots(figsize=(15, 6))

x = np.arange(b[0], b[99], 0.001)
y = np.arange(0.001, 0.6, 0.001)
xx, yy = np.meshgrid(x, y, sparse=True)
z = lnL2((xx, yy))
cs = plt.contourf(x, y, z, 300, cmap='Blues')
#plt.clabel(cs, colors='k', inline=False)
plt.colorbar()
```

Out[218]: <matplotlib.colorbar.Colorbar at 0x4ce0de10>



In [266]:
```
plt.subplots(figsize=(15, 6))

x = np.arange(11.948, 11.95, 0.00001)
y = np.arange(0.452, 0.454, 0.00001)
xx, yy = np.meshgrid(x, y, sparse=True)
z = lnL2((xx, yy))
cs = plt.contourf(x, y, z, 500, cmap='rainbow')
#plt.clabel(cs, colors='k', inline=False)
plt.colorbar()
```

Out[266]: <matplotlib.colorbar.Colorbar at 0x53116f70>



In [267]:
```
m = np.amax(z)
m
```

Out[267]: 4.237588603997082e-11

In [268]:
```
x_min, y_min = 0, 0
for i, j in enumerate(z):
    for k, l in enumerate(j):
        if l == m:
            x_min, y_min = i, k
```

In [269]:
```
z[x_min][y_min], x[x_min], y[y_min]
```

Out[269]:
```
(4.237588603997082e-11, 11.948849999999968, 0.45284000000000085)
```

In [270]:
```
res.x
```

Out[270]:
```
array([11.94884318,  0.45284762])
```

In [ ]: