__Elements of Statistics and Econometrics__

Assignment 1

Problem 3: Inferential Statistics

Import necessary libraries

In [370]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns

%matplotlib inline
```

1. We start with the veri cation of the law of large numbers. Thus we check if an estimator converges (in probability) to its true value if the sample size increases.

(a) Simulate samples of size n = 100, ..., 100000 (with step say 1000) from a normal distribution with mean 1 and variance 1, i.e. $\sim N(1; 1)$. For each sample estimate the mean, the variance. Plot the path of sample means as function of n. What conclusion can we draw from the figure if we keep in mind the law of large numbers?

(b) Frequently if is diffcult to obtain more data. How many observations do we need in order to obtain an estimator which is close enough ± 0.01 to the true value?

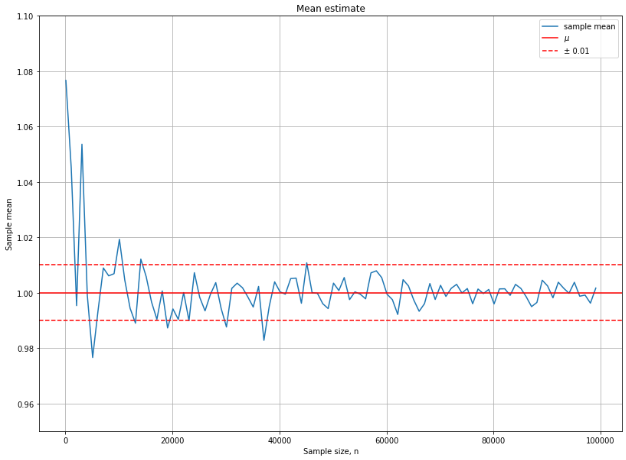
In [106]:

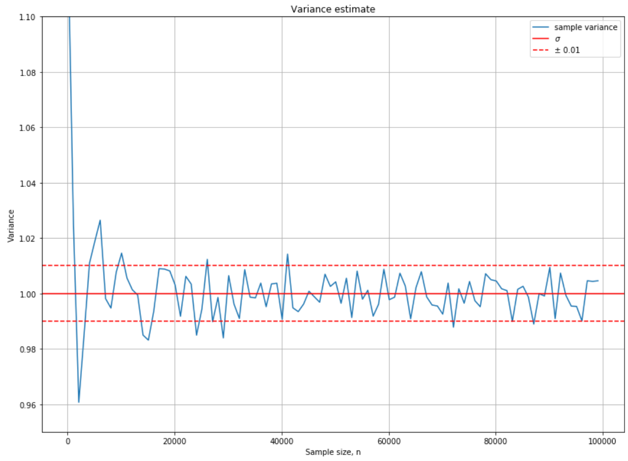
```
mu = 1.
sigma = 1.

n = np.arange(100, 100000, 1000)
samples = np.array([
    np.random.normal(mu, sigma, ni) for ni in n
])

means = np.array([np.mean(sample) for sample in samples])
variances = np.array([np.var(sample) for sample in samples])

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(30, 10))

ax1.plot(n, means, label = 'sample mean')
ax1.axhline(y = mu, color='r', label='$\mu$')
ax1.axhline(y = mu + 0.01, color = 'r', linestyle='dashed', label='± 0.01')
ax1.axhline(y = mu - 0.01, color = 'r', linestyle='dashed')
ax1.set_title("Mean estimate")
ax1.set_xlabel('Sample size, n')
ax1.set_ylabel('Sample mean')
ax1.set_ylim([0.95, 1.1])
ax1.grid()
ax1.legend()

ax2.plot(n, variances, label='sample variance')
ax2.axhline(y = sigma, color='r', label='$\sigma$')
ax2.axhline(y = sigma + 0.01, color = 'r', linestyle='dashed', label='± 0.01')
ax2.axhline(y = sigma - 0.01, color = 'r', linestyle='dashed')
ax2.set_title("Variance estimate")
ax2.set_xlabel('Sample size, n')
ax2.set_ylabel('Variance')
ax2.set_ylim([0.95, 1.1])
ax2.grid()
```

```
ax2.legend()
```

Out[106]:

```
<matplotlib.legend.Legend at 0x117625810>
```





We can conclude, that as $n \to \inf \implies \mu, \sigma \to 1$ which confirms the law of large numbers that the distribution will come to normal with increasing size of sample.

After $n > 25000$, fluctuations of mean became stable($\mu^{'}_{n} \in \begin{bmatrix}\mu - 0.5; \mu + 0.5\end{bmatrix}$, where $\mu^{'}_{n}$ is a mean of generated sample of size n)

After $n > 43000$, fluctuations of variance became stable($\sigma^{'}_{n} \in \begin{bmatrix}\sigma - 0.5; \sigma + 0.5\end{bmatrix}$, where $\sigma^{'}_{n}$ is a variance of generated sample of size n)

(c) Add to the plot the 95% confidence intervals. Do it ones with known σ and ones with an estimated. The confidence intervals have to be constructed manually. Provide their interpretation.

In [117]:

```
def confidence_interval(X, alpha, sigma=None):
    n = len(X)
    mean = np.mean(X)

    if sigma is None:
        ci = (np.std(X, ddof=1) * stats.t(n - 1).ppf(1. - alpha / 2.)) / np.sqrt(n)
    else:
        ci = (sigma * stats.norm.ppf(1. - alpha / 2.)) / np.sqrt(n)


    return mean - ci, mean + ci


def ci_sigma_known(X, alpha, sigma):
    n = len(X)
    z = stats.norm.ppf(1. - alpha / 2.)
    mean = np.mean(X)

    ci = (z * sigma) / np.sqrt(n)
    return mean - ci, mean + ci

def ci_sigma_unknown(X, alpha):
    n = len(X)
    mean = np.mean(X)
    std = np.std(X, ddof=1)

    ci = (std * stats.t(n - 1).ppf(1. - alpha / 2.)) / np.sqrt(n)

    return mean - ci, mean + ci

print(ci_sigma_known(samples[10], 0.05, 1.))
print(ci_sigma_unknown(samples[10], 0.05))

print(confidence_interval(samples[10], 0.05, 1.))
print(confidence_interval(samples[10], 0.05))
```

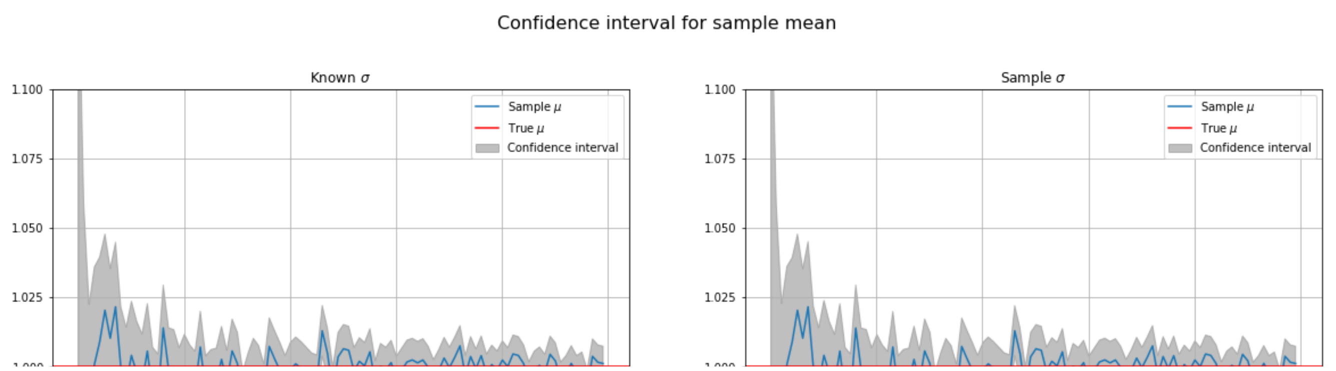
```
(0.9764672869990821, 1.0154720281208722)
(0.9762861569060284, 1.015653158213926)
(0.9764672869990821, 1.0154720281208722)
(0.9762861569060284, 1.015653158213926)
```

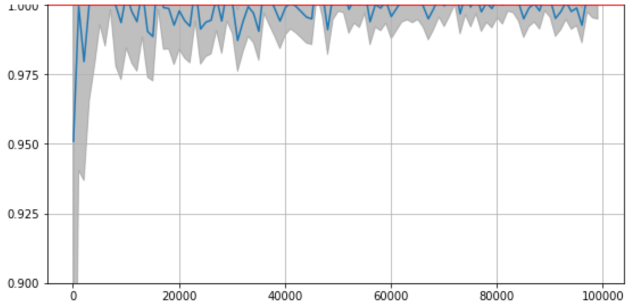
In [133]:

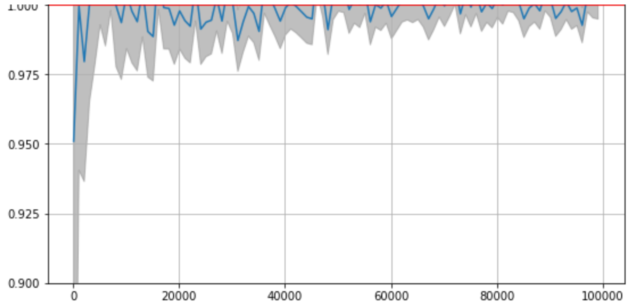
```
mu = 1.
sigma = 1.
alpha = 0.05

n = np.arange(100, 100000, 1000)
samples = np.array([
    np.random.normal(mu, sigma, ni) for ni in n
])

means = np.array([np.mean(sample) for sample in samples])

samples_len = len(samples)

ci_known_min_intervals = np.empty(sample_len)
ci_known_max_intervals = np.empty(sample_len)
ci_unknown_min_intervals = np.empty(sample_len)
ci_unknown_max_intervals = np.empty(sample_len)

for i in range(sample_len):
    ci_known_min_intervals[i], ci_known_max_intervals[i] = confidence_interval(samples[i], alpha, s
igma)
    ci_unknown_min_intervals[i], ci_unknown_max_intervals[i] = confidence_interval(samples[i], alph
a)

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20, 9))
fig.suptitle("Confidence interval for sample mean", fontsize=16)

ax1.set_title("Known $\sigma$")
ax2.set_title("Sample $\sigma$")
ax1.plot(n, means, label = 'Sample $\mu$')
ax2.plot(n, means, label = 'Sample $\mu$')

ax1.fill_between(n, ci_known_min_intervals, ci_known_max_intervals, alpha=0.5, color='gray', label=
'Confidence interval')
ax2.fill_between(n, ci_unknown_min_intervals, ci_unknown_max_intervals, alpha=0.5, color='gray', la
bel='Confidence interval')

ax1.axhline(y = mu, color='r', label='True $\mu$')
ax2.axhline(y = mu, color='r', label='True $\mu$')

ax1.set_ylim([0.90, 1.1])
ax2.set_ylim([0.90, 1.1])


ax1.grid()
ax2.grid()

ax1.legend()
ax2.legend()
```

Out[133]:

```
<matplotlib.legend.Legend at 0x11e647b50>
```







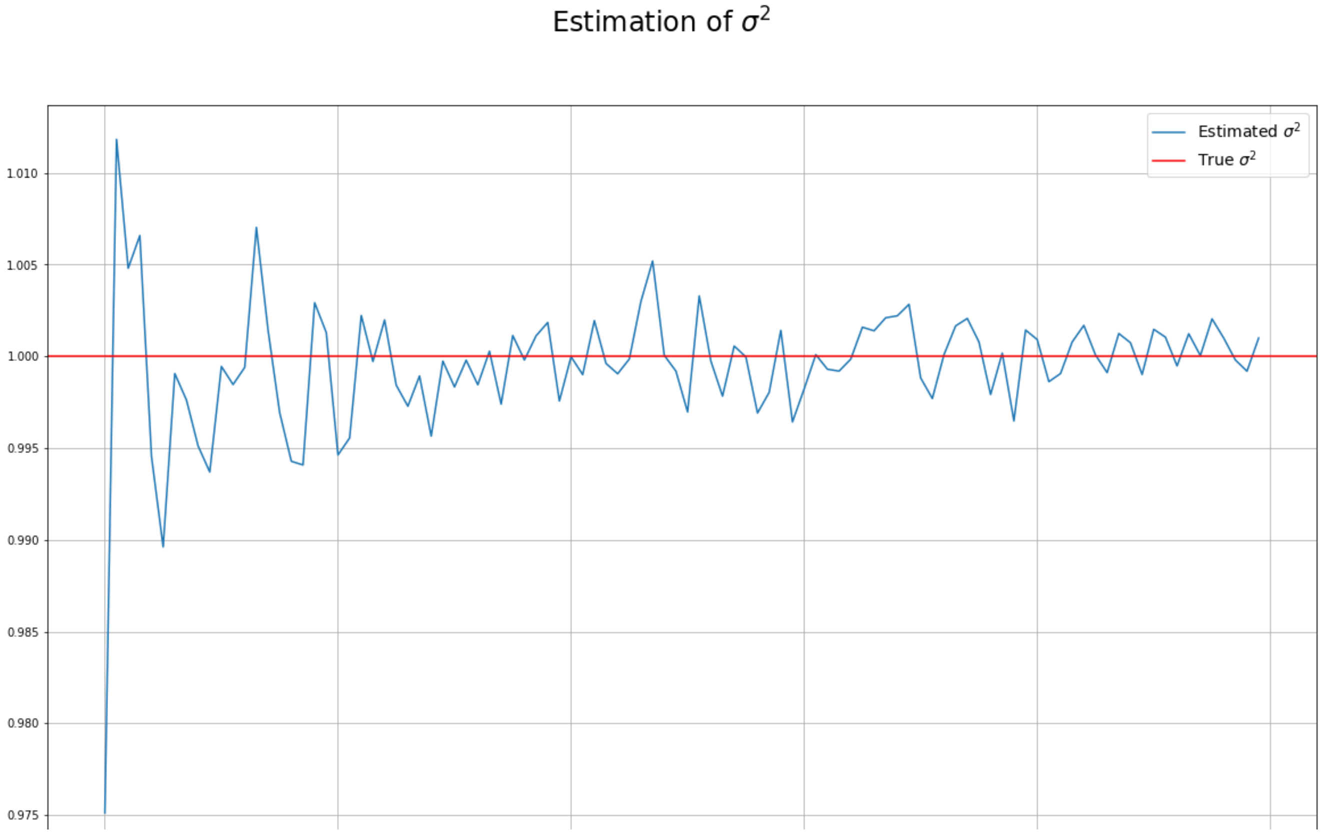
(d) Next plot the sample variance as a function of n. If we think about the consistency of the sample variance as an estimator of σ^2, does the figure support this property? Give the interpretation of consistency in your own words.

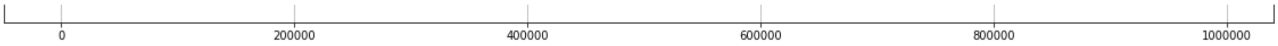
In [153]:

```
sigma = 1.

def sample_variance(n, mu=1., sigma=1.):
    X = np.random.normal(mu, sigma, n)

    return np.var(X, ddof=1)

n = np.arange(100, 1000000, 10000)
samples = np.array([
    np.random.normal(mu, sigma, ni) for ni in n
])

variances = np.array([sample_variance(ni) for ni in n])
```

In [158]:

```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(20, 12))

fig.suptitle("Estimation of $\sigma^2$", fontsize=24)

ax.plot(n, variances, label='Estimated $\sigma^2$')
ax.axhline(y = sigma, color='r', label='True $\sigma^2$')
ax.grid()
ax.legend(fontsize=14)
plt.show()
```





2. The objective of this part is to get a better feeling for the ML estimation procedures. The estimation for non-standard distributions/models usually follows the maximum-likelihood principle. The t-distribution is a popular alternative if the sample distribution is symmetric, but exhibits heavier tails compared to the normal distribution.

(a) Let $x_1, \dots, x_n$ be a given sample. We assume that it stems from a t-distribution with an unknown number of degrees of freedom. Write down the corresponding log-likelihood function. The density function of the t_df-distribution is given by

$$f(x) = \frac{(1 + \frac{x^2}{df})^{- \frac{df + 1}{2}}}{B(df/2, 1/2)\sqrt{df}}$$

where $B(\cdot, \cdot)$ is the beta function **(beta (a, b) in R)**

Answer: $$L(x) = \prod_{i=1}^{n}f(x_i)$$ $$ ln(L(x)) = ln(\prod_{i=1}^{n}f(x_i)) = ln(\prod_{i=1}^{n}\frac{(1+ \frac{x_i^2}{df})^{−\frac{df+1}{2}}}{B(\frac{df}{2}, \frac{1}{2}) \sqrt{df}}) = \sum_{i=1}^{n}ln(\frac{(1+ \frac{x_i^2}{df})^{−\frac{df+1}{2}}}{B(\frac{df}{2}, \frac{1}{2}) \sqrt{df}}) = −\frac{df+1}{2} \sum_{i=1}^{n} (ln(1+ \frac{x_i^2}{df})) - n * ln(B(\frac{df}{2}, \frac{1}{2}) \sqrt{df}) $$

In [190]:

```
def L(df):
    k = - (df + 1)/2

    left_sum = np.sum(np.array([np.log(1. + xi**2 / df) for xi in x]))

    right_sum = - n * np.log(scipy.special.beta(df/2., 0.5) * np.sqrt(df))

    return k * left_sum + right_sum
```

(b) Simulate a sample $n = 100$ from t_5 Maximize the log-likelihood function (numerically) for the given sample and obtain the ML estimator of the number of degrees of freedom. Compare the estimator with the true value.

In [207]:

```
df = 5

n = 100
x = np.random.standard_t(df = 5, size=n)
```

In [208]:

```
estimated = scipy.optimize.minimize(lambda df: -L(df), 0, method='Powell')
estimated
```

```
/Users/ilyakachko/.vcub/lib/python3.7/site-packages/ipykernel_launcher.py:4: RuntimeWarning:
divide by zero encountered in true_divide
  after removing the cwd from sys.path.
/Users/ilyakachko/.vcub/lib/python3.7/site-packages/ipykernel_launcher.py:6: RuntimeWarning:
invalid value encountered in multiply
```

Out[208]:

```
   direc: array([[1.]])
     fun: array(181.03832522)
 message: 'Optimization terminated successfully.'
    nfev: 24
     nit: 2
  status: 0
 success: True
       x: array(2.92000512)
```

With sample of size 100:

Estimated value is 2.92 True value is 5

(c) Increase the sample to $n = 5000$ and compare the new estimator with the true value. Which property of the estimator we expect to observe?

In [209]:

```
n = 5000
x = np.random.standard_t(df = 5, size=n)

estimated = scipy.optimize.minimize(lambda df: -L(df), 0, method='Powell')
estimated
```

```
/Users/ilyakachko/.vcub/lib/python3.7/site-packages/ipykernel_launcher.py:4: RuntimeWarning:
divide by zero encountered in true_divide
  after removing the cwd from sys.path.
/Users/ilyakachko/.vcub/lib/python3.7/site-packages/ipykernel_launcher.py:6: RuntimeWarning:
invalid value encountered in multiply
```

Out[209]:

```
   direc: array([[1.]])
     fun: array(8203.56495595)
 message: 'Optimization terminated successfully.'
    nfev: 27
     nit: 2
  status: 0
 success: True
       x: array(5.01860135)
```

With sample of size 5000:

Estimated value is 5.01 True value is 5

We expect estimated value to be closer to true value as size of sample become larger. And with this sample it is.

5.01 from $n = 5000$ is much closer to 5 than 2.92 from $n = 100$

3. Next we assess the asymptotic distribution of estimators (in the sense of the central limit theorem).

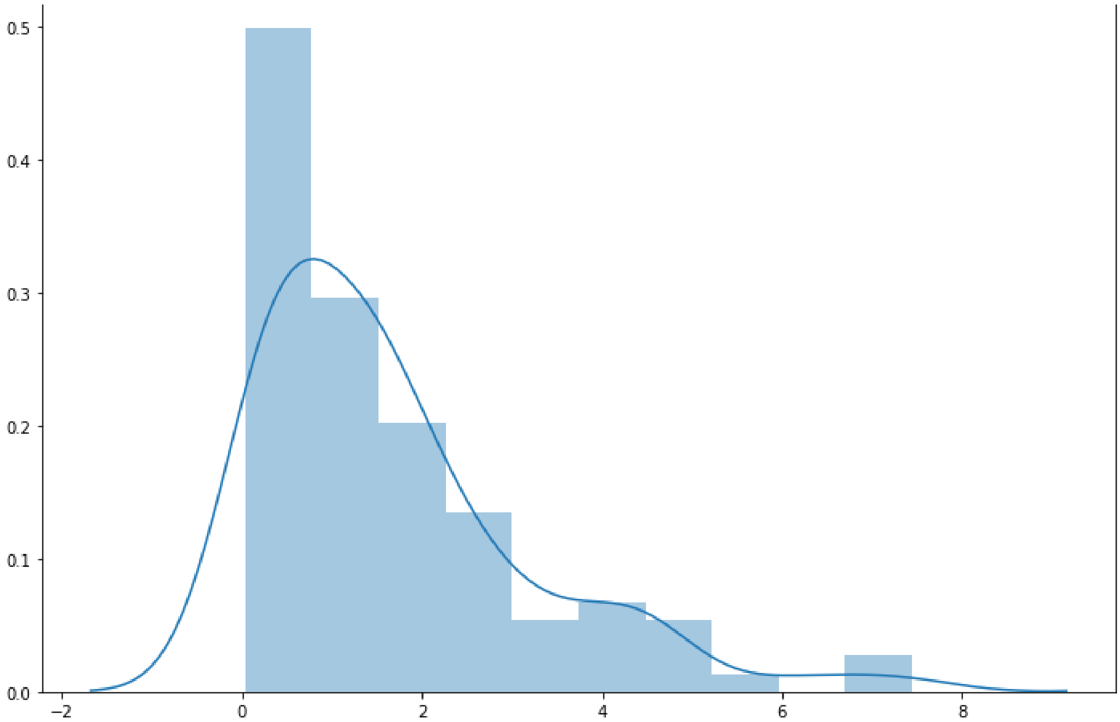
(a) Simulate $b = 1000$ of size $n = 100$ from χ^2_2 distribution with mean 0 and variance 1, i.e. $N(0, 1)$. For each sample estimate the mean, the variance and keep them. Plot the histogram for one of the sample, so that you get a better feeling how the original distribution looks like.

In [300]:

```
size = 100
b = 1000
df = 2
samples = np.array([np.random.chisquare(df=df, size=size) for _ in range(b)])
```

In [301]:

```
fig, ax = plt.subplots()
fig.set_size_inches(12, 8)
fig.suptitle("$\chi^2_2$ Sample distribution of $n = 100$", fontsize=20)
sns.distplot(samples[10]);
```



(b) Plot the KDE or a histogram for the sample of means and the sample of variances. Add the density of the normal distribution for comparison purposes. Compare the density estimator with the normal density. What do you expect and why (statistical reasoning!)?

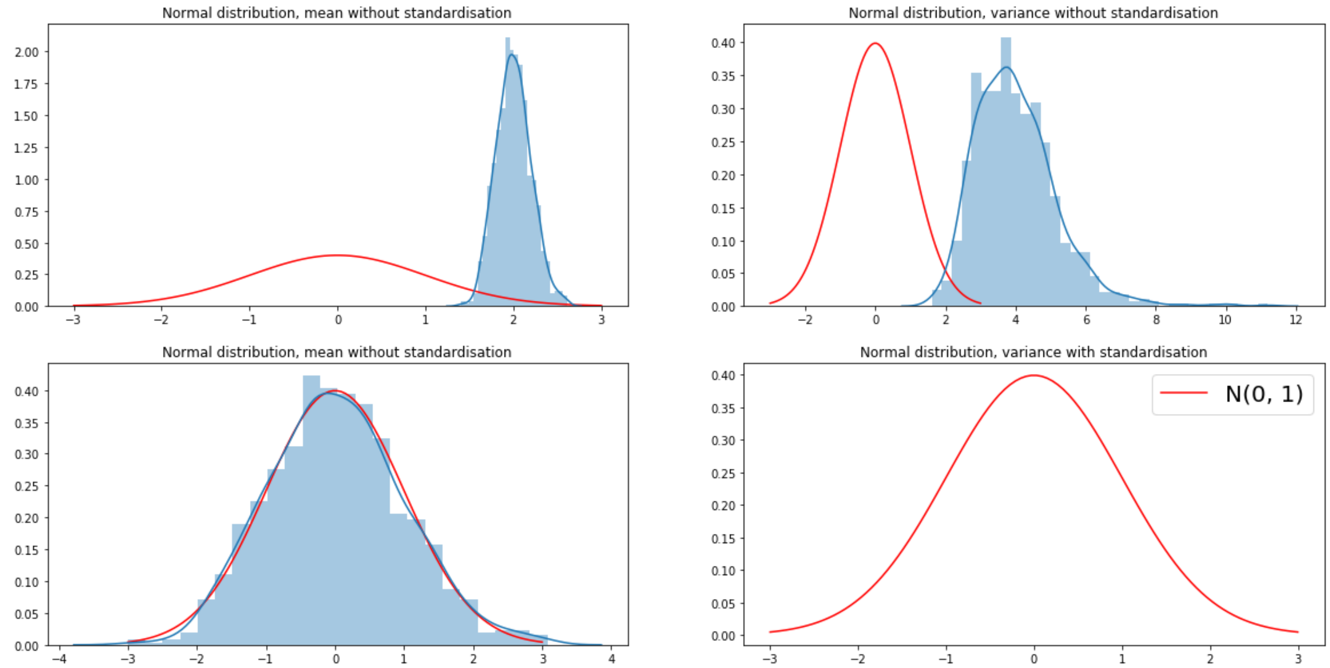
In [302]:

```
mu = 2.
variance = 4.
sigma = np.sqrt(variance)

means = np.empty(b)
variances = np.empty(b)

for i in range(b):
    means[i] = np.mean(samples[i])
    variances[i] = np.var(samples[i])

standard_means = ((means - mu) * np.sqrt(size)) / sigma
```

In [319]:

```
mu = 0
variance = 1.

sigma = np.sqrt(variance)

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))

x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
y = stats.norm.pdf(x, mu, sigma)


ax1.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(means, label='Mean Distribution', ax=ax1)
ax1.set_title("Normal distribution, mean without standardisation")

ax2.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(variances, label='Variance Distribution', ax=ax2)
ax2.set_title("Normal distribution, variance without standardisation")

ax3.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(standard_means, label='Standard Mean Distribution', ax=ax3)
ax3.set_title("Normal distribution, mean without standardisation")

ax4.plot(x, y, color='red', label='N(0, 1)')
ax4.set_title("Normal distribution, variance with standardisation")
```

```
plt.legend(fontsize=20);
```



(c) In the lecture we discussed the CLT for the sample mean. Here it seems to apply to the sample variance too. Why?

In [ ]:

```

```

(d) Let n take values 10^3, 10^4, 10^5 and 10^6. Check the impact of n on the results. Can the statement of the CLT be confirmed?

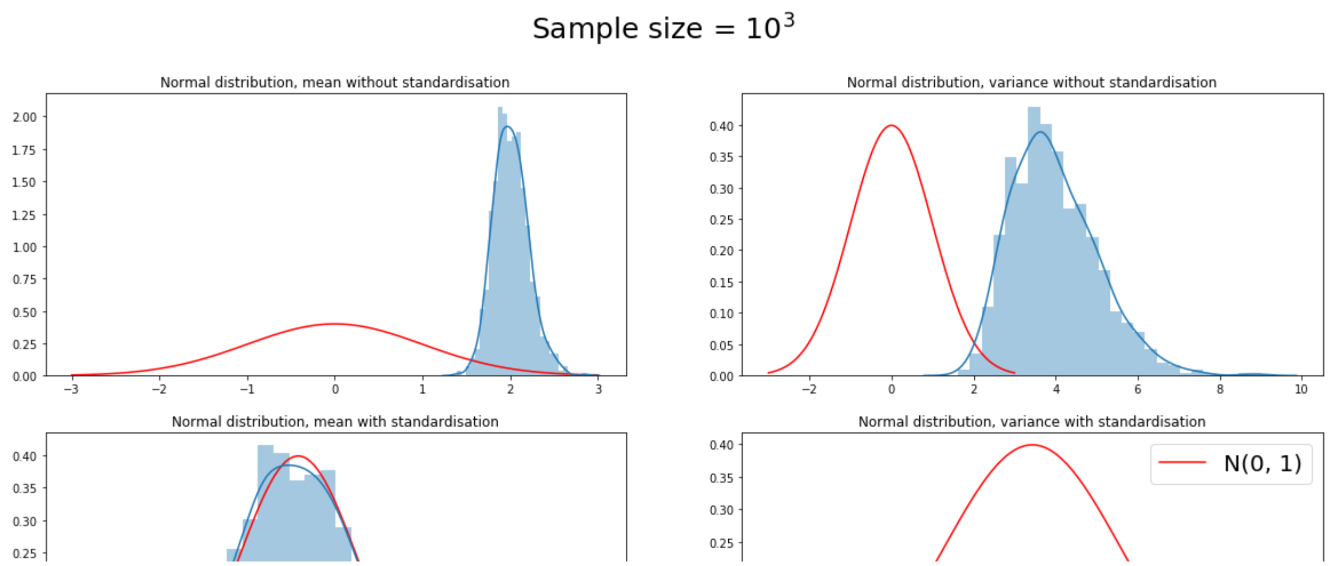
In [310]:

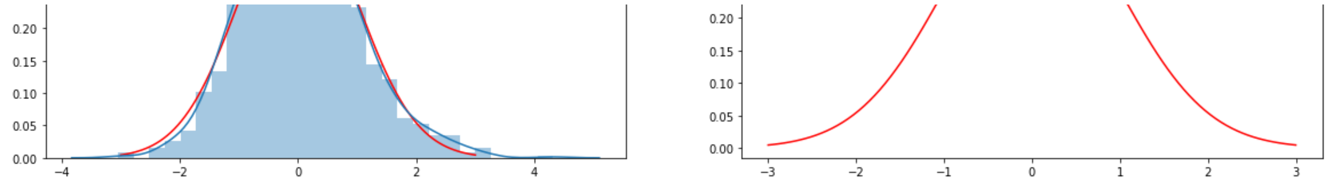
```
size = 100
b_3 = 10**3
b_4 = 10**4
b_5 = 10**5
b_6 = 10**6

df = 2
samples_3 = np.array([np.random.chisquare(df=df, size=size) for _ in range(b_3)])
samples_4 = np.array([np.random.chisquare(df=df, size=size) for _ in range(b_4)])
samples_5 = np.array([np.random.chisquare(df=df, size=size) for _ in range(b_5)])
samples_6 = np.array([np.random.chisquare(df=df, size=size) for _ in range(b_6)])

mu = 2.
variance = 4.
sigma = np.sqrt(variance)

means_3 = np.empty(b_3)
variances_3 = np.empty(b_3)

means_4 = np.empty(b_4)
variances_4 = np.empty(b_4)

means_5 = np.empty(b_5)
variances_5 = np.empty(b_5)

means_6 = np.empty(b_6)
variances_6 = np.empty(b_6)

for i in range(b_3):
    means_3[i] = np.mean(samples_3[i])
    variances_3[i] = np.var(samples_3[i])
```

```
standard_means_3 = ((means_3 - mu) * np.sqrt(size)) / sigma

for i in range(b_4):
    means_4[i] = np.mean(samples_4[i])
    variances_4[i] = np.var(samples_4[i])

standard_means_4 = ((means_4 - mu) * np.sqrt(size)) / sigma


for i in range(b_5):
    means_5[i] = np.mean(samples_5[i])
    variances_5[i] = np.var(samples_5[i])

standard_means_5 = ((means_5 - mu) * np.sqrt(size)) / sigma


for i in range(b_6):
    means_6[i] = np.mean(samples_6[i])
    variances_6[i] = np.var(samples_6[i])

standard_means_6 = ((means_6 - mu) * np.sqrt(size)) / sigma
```

In [320]:

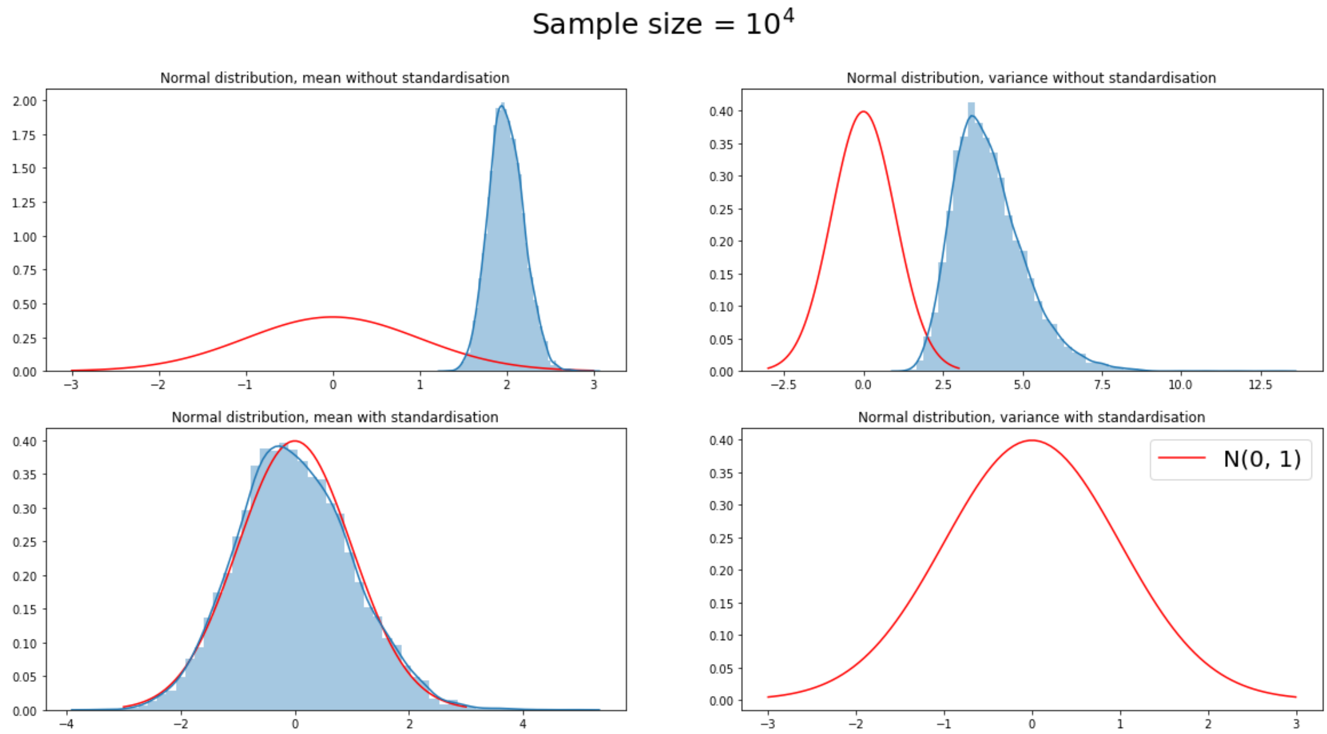
```
mu = 0
variance = 1.

sigma = np.sqrt(variance)

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))

fig.suptitle("Sample size = $10^3$", fontsize=25)

x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
y = stats.norm.pdf(x, mu, sigma)


ax1.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(means_3, label='Mean Distribution', ax=ax1)
ax1.set_title("Normal distribution, mean without standardisation")

ax2.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(variances_3, label='Variance Distribution', ax=ax2)
ax2.set_title("Normal distribution, variance without standardisation")

ax3.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(standard_means_3, label='Standard Mean Distribution', ax=ax3)
ax3.set_title("Normal distribution, mean with standardisation")

ax4.plot(x, y, color='red', label='N(0, 1)')
ax4.set_title("Normal distribution, variance with standardisation")

plt.legend(fontsize=20);
```





In [321]:

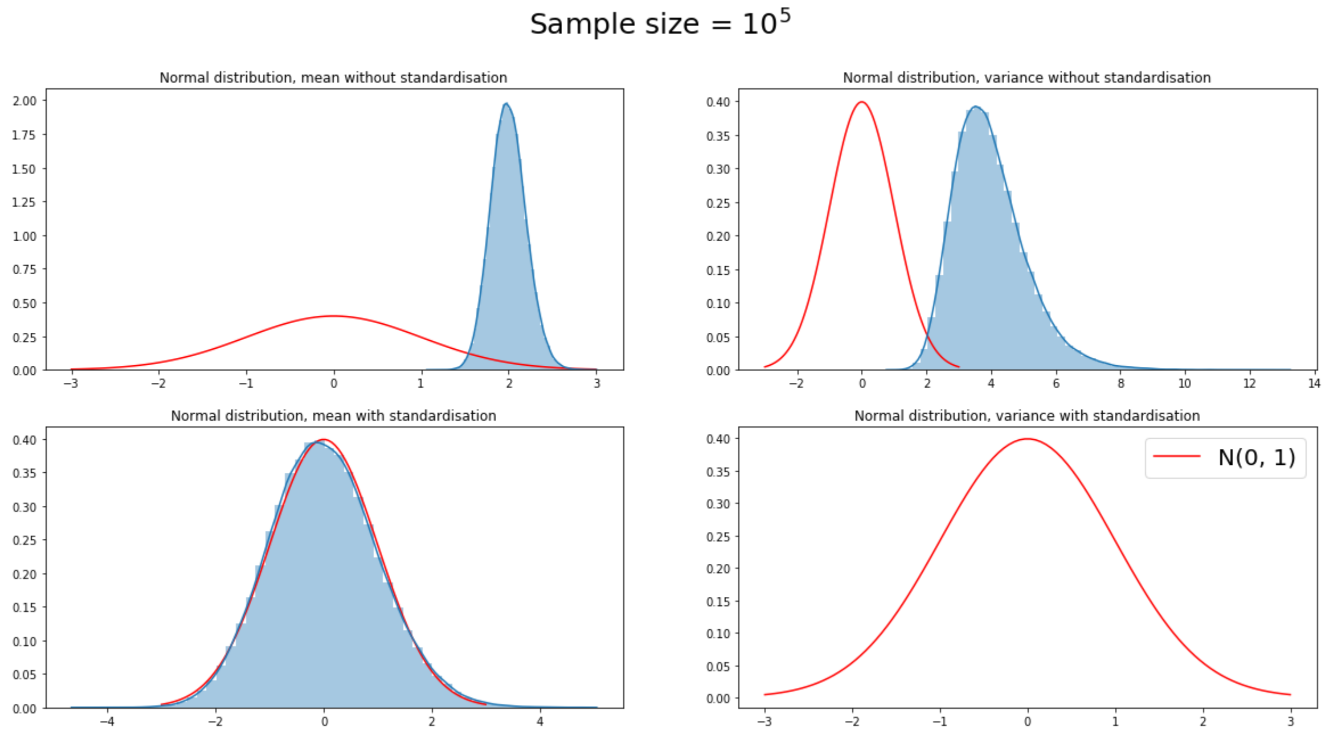
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))

fig.suptitle("Sample size = $10^4$", fontsize=25)

x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
y = stats.norm.pdf(x, mu, sigma)


ax1.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(means_4, label='Mean Distribution', ax=ax1)
ax1.set_title("Normal distribution, mean without standardisation")

ax2.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(variances_4, label='Variance Distribution', ax=ax2)
ax2.set_title("Normal distribution, variance without standardisation")

ax3.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(standard_means_4, label='Standard Mean Distribution', ax=ax3)
ax3.set_title("Normal distribution, mean with standardisation")

ax4.plot(x, y, color='red', label='N(0, 1)')
ax4.set_title("Normal distribution, variance with standardisation")

plt.legend(fontsize=20);
```



In [323]:

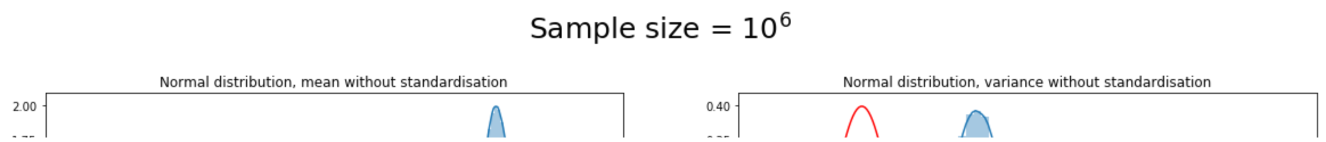
```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))

fig.suptitle("Sample size = $10^5$", fontsize=25)

x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
y = stats.norm.pdf(x, mu, sigma)


ax1.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(means_5, label='Mean Distribution', ax=ax1)
```

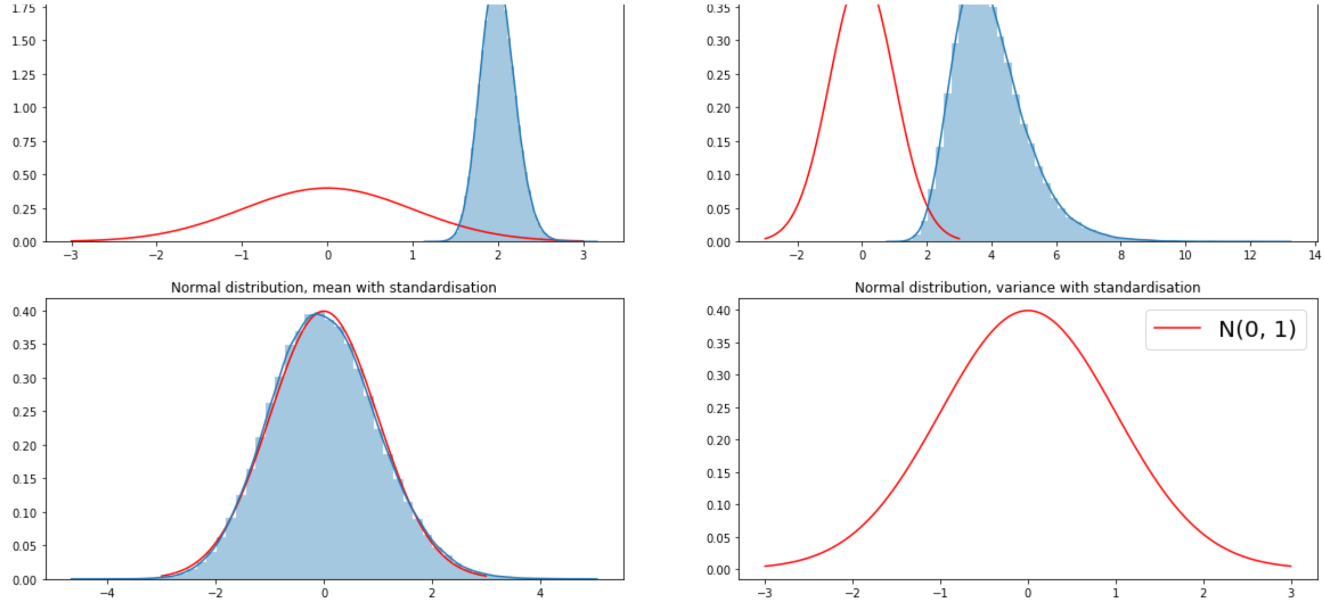
```
ax1.set_title("Normal distribution, mean without standardisation")

ax2.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(variances_5, label='Variance Distribution', ax=ax2)
ax2.set_title("Normal distribution, variance without standardisation")

ax3.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(standard_means_5, label='Standard Mean Distribution', ax=ax3)
ax3.set_title("Normal distribution, mean with standardisation")

ax4.plot(x, y, color='red', label='N(0, 1)')
ax4.set_title("Normal distribution, variance with standardisation")

plt.legend(fontsize=20);
```



In [324]:

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))

fig.suptitle("Sample size = $10^6$", fontsize=25)

x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
y = stats.norm.pdf(x, mu, sigma)


ax1.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(means_6, label='Mean Distribution', ax=ax1)
ax1.set_title("Normal distribution, mean without standardisation")

ax2.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(variances_5, label='Variance Distribution', ax=ax2)
ax2.set_title("Normal distribution, variance without standardisation")

ax3.plot(x, y, color='red', label='N(0, 1)')
sns.distplot(standard_means_5, label='Standard Mean Distribution', ax=ax3)
ax3.set_title("Normal distribution, mean with standardisation")

ax4.plot(x, y, color='red', label='N(0, 1)')
ax4.set_title("Normal distribution, variance with standardisation")

plt.legend(fontsize=20);
```





As sample size become larger, distribution of standardized mean become closer and closer to normal $N(0,1)$ distribution.

4. The next aim objective is check if the probability of type 1 error (size of a test) is correctly attained by a simple two-sided test for the mean.

(a) Simulate a sample of length $n = 100$ from a normal distribution with mean $\mu_0 = 500$ and variance $\sigma^2 = 50$. (Note: you may use the transformation $X = \mu + \sigma Z$, where $Z \sim N(0; 1)$) The objective is to test the null hypothesis $H^0 : \mu = 500$. Assume that σ^2 has to be estimated.

- Compute the test statistics using the formulas in the lecture;
- determine the rejection area for $\alpha = 0.04$ and decide if H_0 can to be rejected.

In [413]:

```
n = 100
mu_0 = 500
variance = 50

sigma = np.sqrt(variance)

alpha = 0.04

sample = np.random.normal(size=n)

X = mu_0 + sigma * sample

X_mean = np.mean(X)
n = len(X)

X_variance = np.sum((X - X_mean)**2)/(n - 1)

print("Estimated sigma^2 = {:.2f}".format(X_variance))
```

```
Estimated sigma^2 = 60.54
```

In [476]:

```
v = (np.abs(np.mean(X) - mu_0) * np.sqrt(n)) / variance
print("v is : {}".format(v))
```

```
v is : 0.1688743153518999
```

In [477]:

```python
z = scipy.stats.norm.ppf(1 - alpha/2.)
print('z value is : {}'.format(z))
```

```
z value is : 2.0537489106318225
```

$$v = (\begin{vmatrix}{\bar{X_n} - \mu}\end{vmatrix} * \sqrt{n})/\sigma = 0.17$$$$N(0; 1): z_{1 - \alpha/2} = z_{1 - 0.02} = z_{0.098} = 2.054 \implies B = (-\infty; -2.054) \cup (2.054; \infty)$$

$v \notin B \implies H_0$ not rejected

With significance level of 4% we can say, that μ will be 500

(b) Determine the p-values using the formulas from the lecture and compare/check the results using a build-in function for this test in R or Python. Give a verbal interpretation of the obtained p-value.

$$Φ(v) = \frac{\alpha^{'}}{2} \implies \alpha^{'} = 2 Φ(v)$$

In [426]:

```python
alpha_hat = (1 - scipy.stats.norm.cdf(v))*2
print(
    "With {}% confidence we can say, that our sample is in correct distribution. p value:  {}".for
mat(int(alpha_hat * 100), alpha_hat)
)
```

```
With 86% confidence we can say, that our sample is in correct distribution. p value:
0.8658955023910901
```

In [459]:

```python
t_stat, p_value = scipy.stats.ttest_1samp(X, 500)

print("p value from build-in library: {}".format(1 - p_value))
print("t_stat value : {}".format(t_stat))
```

```
p value from build-in library: 0.7195435717442351
t_stat value : -1.0852276389477666
```

(c) Simulate $M = 1000$ samples of size $n = 100$ and with $\mu_0 = 500$ and variance $\sigma^2 = 50$. For each sample i run the test (using a standard function) and set $p_i = 0$ if H_0 is not rejected and $p_i = 1$ is rejected. Compute $\hat{\alpha} = \frac{1}{M} \sum_{i = 1}^{M} p_i$. $\hat{\alpha}$ is the empirical confidence level (empirical size) of the test. Compare $\hat{\alpha}$ with α. Do you expect the difference to be large or small and why? Relate it to the assumptions of the test.

In [471]:

```python
M = 1000
n = 100
mu_0 = 500
variance = 50
sigma = np.sqrt(variance)

samples = np.array([np.random.normal(loc=mu_0, scale=sigma, size=n) for _ in range(M)])
```

In [465]:

```python
scipy.stats.ttest_1samp(samples[12], 500)
```

Out[465]:

```
Ttest_1sampResult(statistic=1.3557062757203215, pvalue=0.17827723134288836)
```

In [473]:

```python
test_results = np.array([scipy.stats.ttest_1samp(sample_i, mu_0) for sample_i in samples])

alpha_hat = np.sum(np.array([1 if x[1] > 0.4 else 0 for x in test_results])) / M
```

```
print("Alpha hat is : {}, Alpha is 0.4. Difference is relatively big, because number of samples is quite low.".format(alpha_hat))
```

```
Alpha hat is : 0.582, Alpha is 0.4. Difference is relatively big, because number of samples is
quite low.
```

(d) Assume now that one of the assumptions is not satised. For example, the data is in fact not normal. Repeat the above analysis, but simulate a sample $z_1, \dots, z_n$ from t-distribution with 3 degrees of freedom. Compute $x_i = 500 + z_i/\sqrt{z_i}$ (Note: This will guarantee the same expectation and the same variance as the above normal distribution.) Recompute a new $\hat{\alpha}$. What do you expect? Relate your answer to the type one error and the underlying assumptions.

In [ ]:

```

```

(e) Power of a test: The first objective is to assess the probability of type 2 error (power of a test) of goodness-of-fit test. Goodness-of-fit tests for the normal distribution are of key importance in statistics, since they allow to verify the distributional assumptions required in many models. Here we check the power of the Kolmogorov-Smirnov test, i.e. is the test capable to detect deviations from normality?

* Simulate $M = 1000$ samples of size 100 from a t-distribution with $df = 2, \dots, 50$ degrees of freedom. For each sample run the Kolmogorov-Smirnov test and count the cases when the H_0 of normality is correctly rejected (for each df). How would you use this quantity to estimate the power of the test?

Make an appropriate plot with the df on the X-axis. (Note: the t-distribution converges to the normal distribution as df tends to in nity. For df > 50 the distributions are almost identical.) Discuss the plot and draw conclusions about the reliability of the test.

In [ ]:

```

```