Project 2

author: Alexander Krasovskiy

December 17, 2022

0.0.1 Task 2.1 Documentation

1. Data Source:

- The data source for the Task 2.1 is a generated random uniformly distributed sample of size n (scipy.stats.uniform module is used)
- Initial assumptions: X is a uniformly distributed sample with values in range $[0, \theta]$ (where θ is a model parameter)

2. Research:

1. Test data (generated):

- $X \in$ { X1, X2 , ..., X100 }
- Size of the generated data: |X1| = 1000, |X2| = 2000, ... , |X100| = 100000

2. Possible θ estimators:

- Min(X) = $X_{1:n}$ - Sample X minimum
- Mean(X) = $(X_{1:n}, X_{n:n}) - - - || - - -$ mean
- Max(X) = $X_{n:n} - - - || - - -$ maximum

3. Parameter $\theta = 10$ (or any natural number)

4. Expected test result: **Max(X)** is a sufficient statistics for the parameter θ

5. Alternative result: **Mean(X)** or **Min(X)** is a sufficient statistics for the parameter θ

- Research purpose: check if the expected test result is correct
(Show that **Max(x)** is a sufficient estimator)

[48]:
```python
theta = 10
# Define sample (X) generator
def T(theta, sample_size):
    return uniform.rvs(0,theta,sample_size)
```

[49]:
```python
#Define statistics functions
def Mean(sample):
    return sample.mean()

def Min(sample):
    return sample.min()

def Max(sample):
    return sample.max()
```

4. Probability distribution analysis

1. The **Kolmogorov-Smirnov** is a nonparametric test of the equality of continuous (or discontinuous), one-dimensional probability distributions that can be used to compare a sample with a reference probability distribution (one-sample K–S test), or to compare two samples (two-sample K–S test). In essence, the test answers the question "What is the probability that this collection of samples could have been drawn from that probability distribution?"
2. The solution depends on a function scipy.stats.kstest that implements the **Kolmogorov-Smirnov** test.
3. A brief description of the Kolmogorov-Smirnov test:

- method: scipy.stats.kstest(rvs, cdf, args=(), N=20, alternative='two-sided', method='auto')
- reference: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kstest.html
- Performs the (one-sample or two-sample) Kolmogorov-Smirnov test for goodness of fit. The one-sample test compares the underlying distribution F(x) of a sample against a given distribution G(x) **(this option is used)**. The two-sample test compares the underlying distributions of two independent samples. Both tests are valid only for continuous distributions.
- parameters:
  - rvs - random values $\equiv X$
  - cdf - expected cumulative distribution function $\equiv \mathrm{Unif}(0, \theta)$
  - N - sample size $\equiv$ size of X
  - If the sample comes from distribution $F(x) \equiv cdf$, then D_n converges to 0 almost surely in the limit when n goes to infinity.
- Let $D_n = \sup_x |F_n(x) - F(x)|$
- H0 (K-S Test): $\sqrt{n}D_n$ converges to the Kolmogorov distribution, which does not depend on the empirical CDF $F(x)$

[50]:
```python
from scipy.stats import kstest
Lambda = 0.05
n = 100000
test = kstest(T(theta,n),uniform.cdf, N=n, args=(0,theta))
s, p_value = test
print("sample size = %d"%n)
print(test)
if (p_value>Lambda):
    print()
    print("H0 for the Kolmogorov-Smirnov test is correct, lambda = %.3f (%d%%␣
↪percentile)"%(Lambda,100-100*Lambda))
    print()
    print("The sample X distribution is likely to be equivalent to␣
↪Unif[0,theta]")
else:
    print("H0 for the Kolmogorov-Smirnov test is not correct, lambda = %.
↪3f"%Lambda)
    print("X distribution is unlikely to be equivalent to Unif[0,theta]")
```

```
sample size = 100000
KstestResult(statistic=0.0026661944797430337, pvalue=0.4750146065722749)

The Kolmogorov-Smirnov test H0 is correct, lambda = 0.050 (95 percentile)

The sample X distribution is equivalent to Unif[0,theta] with probability >= 95%
```

4. As expected, the p-value of 0.47 is not below our threshold of 0.05, so we cannot reject the null hypothesis. =>

- X is a random sample with the distribution = Unif(0,ϑ) with a probability >= 95%

5. Additionally, it is necessary to check, that the sample generated by T is simple.

- function isSimpleRandomSample(T, theta, expected_range, eps = 10^{-5}) - to check if a uniformly distributed random variable T is a simple random sample for the expected range [a,b].
  1. Calculate the expected mean and standard deviation of the Uniform[a,b] distribution
  2. Iterate for each sample size =n min: 10, max: 10^7
  3. For each sample size (=n):
     - Generate a random sample of size n
     - Calculate [a_iter,b_iter] == [min(sample),max(sample)]
     - Compute p == P(x<a_iter or x>b_iter)
     - Calculate the sample mean and standard deviation
     - In order to prove that T(n) is a simple random sample, we need to check that the random variable T(n) is not limited for the population [0,theta]
     - Therefore, it is necessary and sufficient to check that lim(p)=0, n->inf for a given eps.
     - => If abs(p) < eps: T generates a simple random sample.
       * return True
  4. Else: - return False

[57]:
```python
from math import sqrt

def isSimpleRandomSample(T, theta, expected_range, eps):
    a_iter=None
    b_iter=None
    diff = abs(expected_range[0]-expected_range[1])
    sample_sizes = [int(x) for x in [10,1e2,1e3,1e4,1e5,1e6, 1e7]]
    print("expected range:",expected_range)
    expected_mean = (expected_range[1]-expected_range[0])/2
    expected_std = sqrt(1/12 * (expected_range[1]-expected_range[0])**2)
    print("expected mean:",expected_mean)
    print("expected standard deviation:",expected_std)
    print()
    for size in sample_sizes:
        samp = T(theta, size)
        a_iter = samp.min()
        b_iter = samp.max()

        p = abs(a_iter-expected_range[0])/diff + abs(b_iter-expected_range[1])/
 ↪diff
        print("for sample size = %d:"%size)
        print("p_value == P(x<a_iter or x>b_iter) = %.8f"%p)
        print("sample mean = E(X):",samp.mean())
        print("sample standard deviation = sqrt(Var(X)):",samp.std())
        print("actual range: ",[a_iter,b_iter])
        print()

        if p < eps:
            print("Random sample is simple, eps=%e"%eps)
            return True
    return False

eps=1e-5
population=[0,theta]

if isSimpleRandomSample(T,theta,population, eps=eps):
    success("\nT(n) generates a simple random sample for the expected␣
 ↪population=%s, eps=%e"%(population,eps))
else:
    fail("\nT(n) does not generate a simple random sample for the expected␣
 ↪population=%s, eps=%e"%(population,eps))
```

```
expected range: [0, 10]
expected mean: 5.0
expected standard deviation: 2.8867513459481287

for sample size = 10:
p_value == P(x<a_iter or x>b_iter) = 0.10368617
```

```
sample mean = E(X): 5.643999061499329
sample standard deviation = sqrt(Var(X)): 2.871822291800391
actual range:  [0.9269957357854164, 9.890134010618748]

for sample size = 100:
p_value == P(x<a_iter or x>b_iter) = 0.00762663
sample mean = E(X): 4.67245060952827
sample standard deviation = sqrt(Var(X)): 3.058244597728054
actual range:  [0.028974161437055335, 9.952707822710888]

for sample size = 1000:
p_value == P(x<a_iter or x>b_iter) = 0.00154257
sample mean = E(X): 4.990070772551295
sample standard deviation = sqrt(Var(X)): 2.9470834430016635
actual range:  [0.009418768815103729, 9.993993082600863]

for sample size = 10000:
p_value == P(x<a_iter or x>b_iter) = 0.00020790
sample mean = E(X): 5.0136269329335414
sample standard deviation = sqrt(Var(X)): 2.9008435888900697
actual range:  [0.001960486448134846, 9.999881494744043]

for sample size = 100000:
p_value == P(x<a_iter or x>b_iter) = 0.00000881
sample mean = E(X): 4.99221187269252
sample standard deviation = sqrt(Var(X)): 2.8887436061886724
actual range:  [2.3191380660314564e-05, 9.999935093523947]

Random sample is simple, eps=1.000000e-05

T(n) generates a simple random sample for the expected population=[0, 10],
eps=1.000000e-05
```

5. Model definition: $(X, \{P_\theta, \theta \in \Theta\})$:
- X $= [0, \theta]$
- $\theta = 10$; $\Theta = \mathbb{R}$
- $P_\theta \equiv Unif(0, \theta)$

6. Analysis (θ estimation):

1. Compute the mean squared error of $|\theta - S(X)|$

- Where $S(X)$ is a statistics function
- The squared error is computer for every sample $X \in$ { X1, X2 , ..., X100 }: $E_i = |\theta - T(X_i)|$
- => The mean squared error is equal to $\frac{1}{n}(E_1 + E_2 + ... + E_{100})$ = MSE

Brief description of the function **error(theta, stat, T)**:

1. Arg: θ - estimated parameter

2. Arg: $T == T(\theta, n)$ - random sample generator

3. Arg: *stat* - a reference to the statistics function of a sample T(θ, sample size)

4. for each sample size in range(1000,100000, with step 1000):

   - Approximate θ based on the random sample *stat*(T(θ, sample size))

   - Compute the squared error between theta and theta approximation

   - Compute the mean squared error MSE

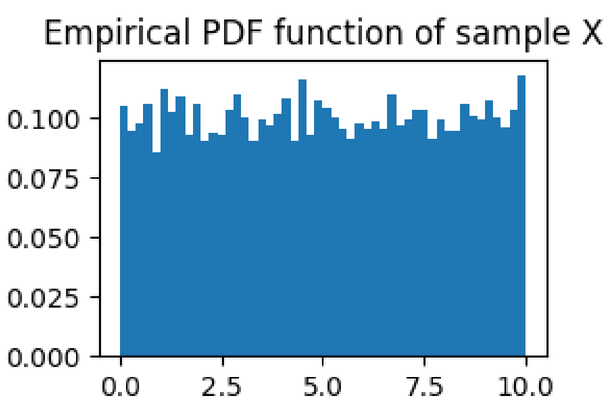
5. Print results & draw the error plot

[67]:
```
def error(theta,stat, T):
    print("Theta =",theta)
    print("Theta estimator:",stat)
    mse = 0
    error_arr=[]
    sample_sizes=range(1000,101000,1000)
    for sample_size in sample_sizes:
        theta_appr = stat(T(theta,sample_size))
        sqe=(theta-theta_appr)**2
        mse+=sqe
        error_arr.append(sqe)
    mse/=len(sample_sizes)
    print("MSE:",mse)
    plt.plot(sample_sizes,error_arr, color="b", label="squared error")
    plt.axhline(mse,color="r", label="mean squared error")
    return mse
```

Empirical PDF Preview

- A method **plt.hist()** is used for plotting the empirical PDF
- **density=True** indicates that the histogram represents a probability density function

[68]:
```
plt.hist(T(theta,10000),bins=50,density=True)
plt.title("Empirical PDF function of sample X")
plt.gcf().set_size_inches(3,2)
plt.show()
```

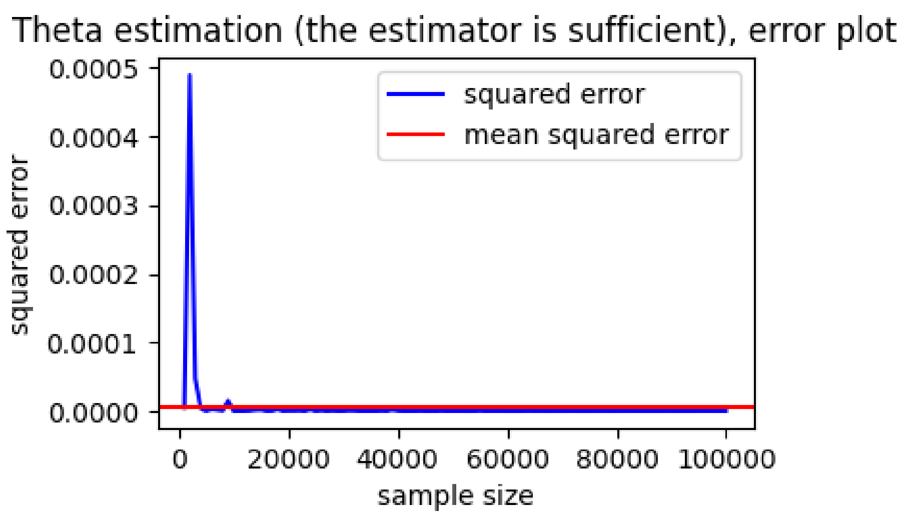


Theta estimation

- In the example below, **error()** function is used for computing the θ estimation MSE (and the SE for each sample size):

[71]:
```
plt.gcf().set_size_inches(4,2.5)
plt.xlabel("sample size")
plt.ylabel("squared error")
mse = error(theta,Max, T)
plt.legend(loc=1)
plt.title("Theta estimation (the estimator is sufficient), error plot")
plt.show()
if (mse<Lambda):
    success("The estimator %s is sufficient."%Max)
else:
    fail("The estimator %s is not sufficient."%Max)
```

```
Theta = 10
Theta estimator: <function Max at 0x000001F5E2675510>
MSE: 5.81649238563254e-06
```

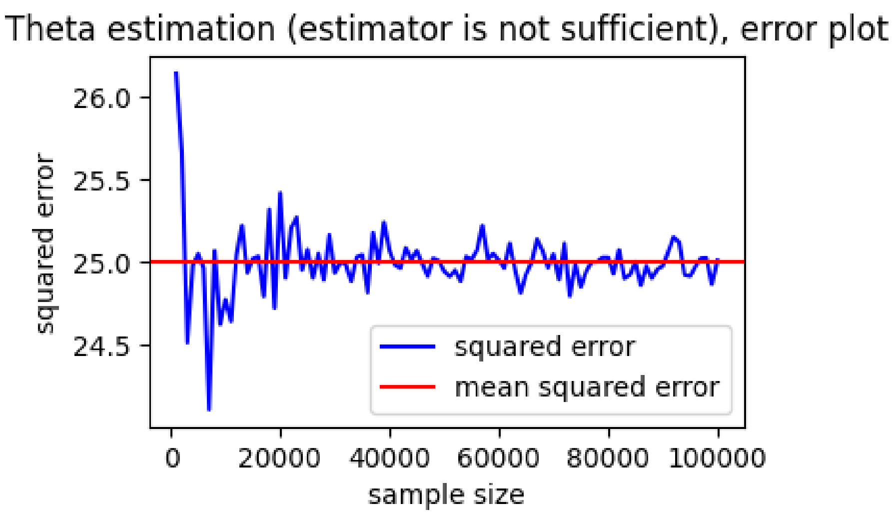



```
The estimator <function Max at 0x000001F5E2675510> is sufficient.
```

- To illustrate the difference between consistent and incinsistent estimators, θ is estimated with a different statistics function $-$ **Mean(X)**, which is not consistent

[72]:
```
plt.gcf().set_size_inches(4,2.5)
plt.xlabel("sample size")
plt.ylabel("squared error")
mse = error(theta,Mean, T)
plt.legend(loc=4)
plt.title("Theta estimation (estimator is not sufficient), error plot")
plt.show()
if (mse<Lambda):
    success("The estimator %s is sufficient."%Mean)
else:
    fail("The estimator %s is not sufficient."%Mean)
```

```
Theta = 10
Theta estimator: <function Mean at 0x000001F5E1EF15A0>
MSE: 25.000415469126352
```




```
The estimator <function Mean at 0x000001F5E1EF15A0> is not sufficient.
```

7. Conclusions

1. Sample distribution analysis conclusions:

- $X \equiv \mathit{Unif}\,[0, \theta]$

2. H0: $\mathbf{Max(X)} \equiv X_{n:n}$ is a sufficient statistics for the parameter
(Max(X) is asymptotically normal for the parameter theta, because the
SE (standard error) $\rightarrow 0, n \rightarrow \infty$
as shown using the function error())

3. It is not possible to reject the **Kolmogorov-Smirnov** test H0, which states that the empirical CDF of X does not come from a family of Uniform distribution: $\mathit{Unif}\,[0, \theta]$, p-value=0.05

4. Random sample generator $T(\theta, n)$ generates simple random samples, that have distribution equivalent to $\mathit{Unif}\,[0, \theta]$

5. Statistical Model:

- $(X, \{P_\theta, \theta \in \Theta\})$
  - X = $[0, \theta]$
  - $\theta = 10$; $\Theta = \mathbb{R}$
  - $P_\theta \equiv \mathit{Unif}(0, \theta)$