Exercise

January 19, 2024

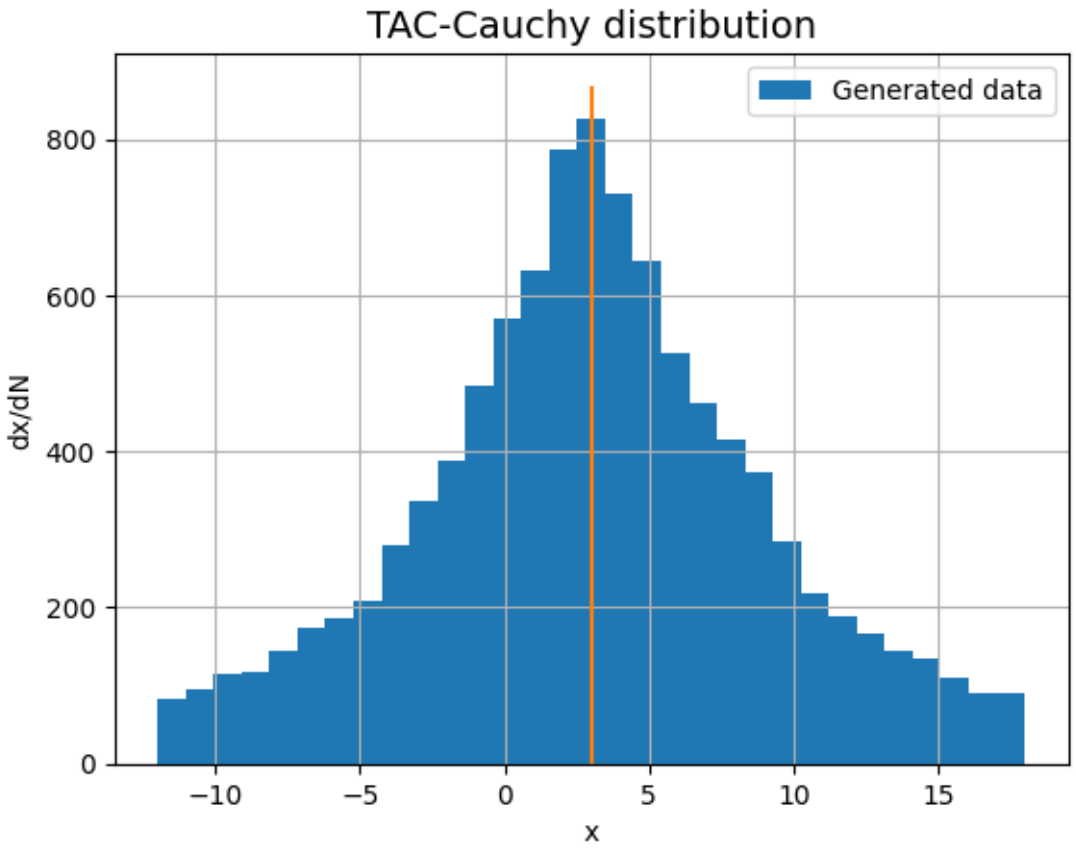
[ ]:
```python
import random
import math

def uniform_range(minimum: float,
                  maximum: float) -> float:
    """
    Generation of a pseudo-casual number distributed accordingly to uniform␣
 ↪distribution between
    [minimum, maximum)

    Args:
        minimum: lower limit of the range (included)
        maximum: upper limit of the range (excluded)

    Returns:
        A pseudo-casual numbers generated according to uniform distribution␣
 ↪between [minimum, maximum)
    """

    return minimum + (random.random() * (maximum - minimum))


def func(x, G, M):
    return (1/math.pi)*(G/((x-M)**2 + G**2))

def tac_cauchy(G: float,
               M: float,
               half_width: float,
               seed: float = 0.) -> float:
    """
    Generation of a pseudo-casual number distributed accordingly to the Cauchy␣
 ↪distribution
    with the try-an-catch algorithm, into a "box" delimited by x_minimum and␣
 ↪x_maximum for the
    horizontal axis and y_minimum and y_maximum for the vertical axis starting␣
 ↪from an optional seed
    different from 0.
```

```
    Args:
        G: parameter of the function
        M: parameter of the function
        half_width: half width of the interval over the horizontal axis
        seed: starting seed for the random generation (optional)

    Returns:
        A single number generated with the try-an-catch algorithm
    """

    if seed != 0.:
        random.seed(seed)
    x = uniform_range(M - half_width, M + half_width)
    y = uniform_range(0, 1/(math.pi*G))

    while y > func(x, G, M):
        x = uniform_range(M - half_width, M + half_width)
        y = uniform_range(0, 1/(math.pi*G))
    return x
```

[ ]:
```
N = 10000
data = []

gamma = 5
M = 3

for i in range(N):
    data.append(tac_cauchy(gamma, M, 3*gamma))
```

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

def sturges(sample: list[float]) -> int:
    """
    Calculation of the optimal number of bins to plot a histogram using the␣
 ↪sturges rule

    Args:
        sample: list of floats representing data

    Returns:
        The number of bins according to sturges rule
    """

    return int(math.ceil(1 + 3.322 * math.log(len(sample))))
```

```
bin_content, bin_edges = np.histogram(data, bins=np.linspace(math.
 ↪floor(min(data)), math.ceil(max(data)), sturges(data)))

fig, ax = plt.subplots(1, 1)
ax.hist(data, bins=bin_edges, label="Generated data")
plt.plot([M, M], ax.get_ylim())
ax.set_title("TAC-Cauchy distribution", size=14)
ax.set_xlabel("x")
ax.set_ylabel("dx/dN")
ax.grid(True)
ax.legend()
plt.show()
```



[ ]:
```
def mean(sample: list[float]) -> float:
    """
    Calculation of the mean of the sample present in the object

    Args:
        sample: list of floats representing data
```

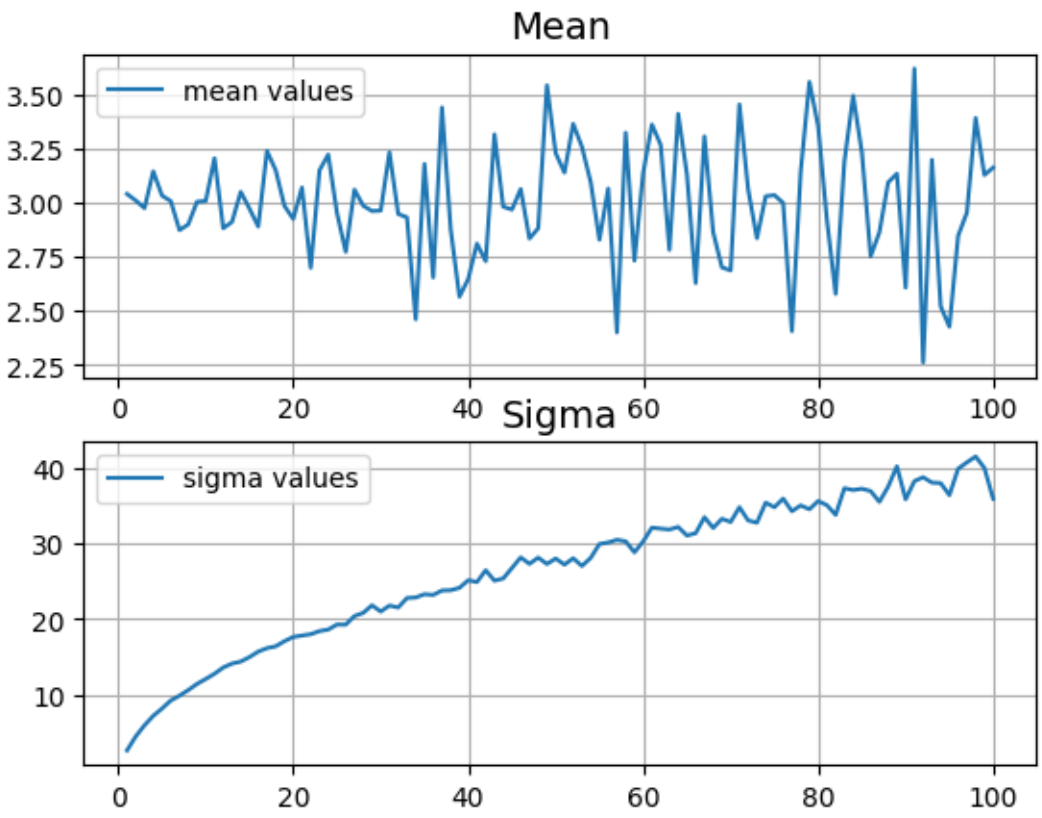
```
    Returns:
        The mean of the sample
    """

    summ = sum(sample)
    n = len(sample)
    return summ / n

def variance(sample: list[float],
             bessel: bool = True) -> float:
    """
    Calculation of the variance of the sample present in the object

    Args:
        sample: list of floats representing data
        bessel: applies the bessel correction (optional, default: True)

    Returns:
        The variance of the sample
    """

    summ = 0.
    sum_sq = 0.
    n = len(sample)
    for elem in sample:
        summ += elem
        sum_sq += elem * elem
    var = sum_sq / n - summ * summ / (n * n)
    if bessel:
        var = n * var / (n - 1)
    return var

def stddev(sample: list[float],
           bessel: bool = True) -> float:
    """
    Calculation of the standard deviation of the sample present in the object

    Args:
        sample: list of floats representing data
        bessel: applies the bessel correction (optional, default: True)

    Returns:
        The standard deviation of the sample
    """

    return math.sqrt(variance(sample, bessel))
```

[ ]:
```
means = []
variances = []

for i in range(100):
    set_data = []
    for j in range(N):
        set_data.append(tac_cauchy(gamma, M, (i+1)*gamma))
    means.append(mean(set_data))
    variances.append(stddev(set_data))

print(means, variances)
```

```
[3.0396507142688765, 3.0092460127270644, 2.9742195515274075, 3.1441614153125603,
3.0320600234839636, 3.006643528129997, 2.872521648077379, 2.899081907039108,
3.003559685694912, 3.0074583218974373, 3.2043818428701596, 2.8813927494388736,
2.9102163606239535, 3.049415296194752, 2.9717239860669444, 2.8891797184025627,
3.2379489743764838, 3.1485513984211564, 2.984206542178195, 2.92449021743393,
3.0696543402638494, 2.698925515907549, 3.148819292941032, 3.2213126223582735,
2.9525813213889234, 2.772219393435811, 3.0596077253513814, 2.9850910136319007,
2.9607798448320306, 2.96286428116385, 3.231559383948999, 2.9472863394495503,
2.931611613789551, 2.4611248478391983, 3.178321182005338, 2.6542881310552926,
3.438891082859325, 2.8751815875487425, 2.565267025587138, 2.6471442985373064,
2.8103781339327942, 2.7291607294161735, 3.314725083817255, 2.980916316961246,
2.967410468114944, 3.0625536468072534, 2.8339430806197035, 2.881203178258506,
3.5410776638276875, 3.2280761033059244, 3.1379901908641474, 3.363446161145516,
3.257044631889516, 3.0942020547370594, 2.828381372062558, 3.0646572591011325,
2.4004342423863365, 3.322469760558047, 2.7319534056387167, 3.1366183465402266,
3.360306959404088, 3.265868885843557, 2.7813539692511817, 3.4094142894280375,
3.1303700140480215, 2.628449355365195, 3.3055164989729193, 2.8633098502054573,
2.700858166355544, 2.686552455174807, 3.452667488458147, 3.059848968993455,
2.8351432449043825, 3.02878622774871, 3.0344562497189598, 2.997789290386229,
2.4064298311392407, 3.141927410860885, 3.556873228242592, 3.352503884660638,
2.9204823882155844, 2.5783540160340213, 3.1909933381512716, 3.492750951776954,
3.230455802087568, 2.7517147575253387, 2.8629071642926647, 3.0931069032752783,
3.1332957153470002, 2.607383499722212, 3.618327104174181, 2.261870806782229,
3.197465637791341, 2.521325137790653, 2.426542081431768, 2.8467027680214176,
2.953503506436611, 3.3904289180788854, 3.1270301731479377, 3.161091549370527]
[2.5886047448555742, 4.444629346700247, 5.921481584553576, 7.168002302878234,
8.138381689515194, 9.199098728156939, 9.845912905741468, 10.583700929745552,
11.40331953852039, 12.060181259722581, 12.749577989525031, 13.58096981941045,
14.099562168105738, 14.364486265810122, 14.960907845346862, 15.689125524696083,
16.14696941810211, 16.39199734452539, 17.074413779468006, 17.63010960168501,
17.81936176679106, 17.995709387201547, 18.406236155496696, 18.610305035461646,
19.278269802257817, 19.26454544156829, 20.402172631094604, 20.820255271429534,
21.831125234562595, 20.997668274014842, 21.79331108679413, 21.531897339345036,
22.793052383461482, 22.862063845594864, 23.298738312865478, 23.205826927041677,
23.79452305894326, 23.831139524005035, 24.143151839706466, 25.149568779604994,
24.91283766232392, 26.473972174186905, 25.09095685040878, 25.385950820963735,
```

```
26.774184537541114, 28.179153161222036, 27.322903806183653, 28.151657204642852,
27.30578875016593, 28.031547445217793, 27.21930909685813, 28.064181013690966,
27.04401258748295, 28.113825413654524, 29.95511101973921, 30.15850514935872,
30.511492489577083, 30.307414288189545, 28.84379444204426, 30.3068623073196,
32.119517940847395, 31.989676825545303, 31.876710138281886, 32.21514065831015,
31.08163613861202, 31.382648392817106, 33.507163752598466, 32.084634348542146,
33.297518765716916, 32.877413496856946, 34.83383138188822, 33.07583854831695,
32.78392717082726, 35.433319536055876, 34.833346205793944, 35.97719775496285,
34.295874752375184, 35.06949490020829, 34.56373562875088, 35.649790983565126,
35.103013798034375, 33.82031204002408, 37.35234611471674, 37.11149127096931,
37.28740941840091, 36.97154033738829, 35.54507839175456, 37.55493958018237,
40.25564393779035, 35.87719791045321, 38.293184314726766, 38.807318057593406,
38.09361772017202, 38.03411155234984, 36.44040847573136, 39.94115312698454,
40.776694911031, 41.556292240691455, 40.03799456273372, 35.9624045359832]
```

[ ]:
```
fig, ax = plt.subplots(nrows=2, ncols=1)
ax[0].errorbar(np.linspace(1, 100, 100), means, label="mean values")
ax[0].set_title("Mean", size=14)
ax[1].errorbar(np.linspace(1, 100, 100), variances, label="sigma values")
ax[1].set_title("Sigma", size=14)
ax[0].grid(True)
ax[1].grid(True)
ax[0].legend()
ax[1].legend()
plt.show()
```



[ ]:
```
def clt_ms_cauchy(G: float,
                  M: float,
                  half_width: float,
                  n_sum: int = 10) -> float:



    y = 0.
    for i in range(n_sum):
        y += tac_cauchy(G, M, half_width)
    y /= n_sum
    return y
```

[ ]:
```
import scipy as sp

clt = []
for t in range(10000):
    clt.append(clt_ms_cauchy(gamma, M, 3*gamma))

print(np.mean(clt))
print(np.std(clt))
print(sp.stats.skew(clt))
```

```
print(sp.stats.kurtosis(clt))
```

```
2.9850832049402607
1.876349798143961
-0.013839731017168521
-0.013650795457602172
```

[ ]:
```
from iminuit import Minuit
from iminuit.cost import UnbinnedNLL

def pdf(x, mu, sigma):
    return sp.stats.norm.pdf(x, mu, sigma)

cost_func = UnbinnedNLL(clt, pdf)
my_minuit = Minuit(cost_func, mu=np.mean(clt), sigma=np.std(clt))
my_minuit.limits['sigma'] = (0, None)

my_minuit.migrad()
my_minuit.minos()
display(my_minuit)
```

Migrad

FCN = 4.097e+04	Nfcn = 59
EDM = 7.56e-15 (Goal: 0.0002)	
Valid Minimum	Below EDM threshold (goal x 10)
No parameters at limit	Below call limit
Hesse ok	Covariance accurate

	Name	Value	Hesse Err	Minos Err-	Minos Err+	Limit-	Limit+	Fixed
0	mu	2.985	0.019	-0.019	0.019			
1	sigma	1.876	0.013	-0.013	0.013	0		

	mu		sigma	
Error	-0.019	0.019	-0.013	0.013
Valid	True	True	True	True
At Limit	False	False	False	False

Max FCN	False	False	False	False
New Min	False	False	False	False

	mu	sigma
mu	0.000352	0
sigma	0	0.000176