Albert-Ludwigs-Universität Freiburg Wintersemester 2022/23

Datenanalyse für Naturwissenschaftler*Innen

Statistische Methoden in Theorie und Praxis

Vorlesung: Dr. Andrea Knue

Übungsleitung: Dr. Constantin Heidegger

Übung 5

Ausgabe: 18. November 2022 10:00 Uhr, Abgabe: 25. November 2022 bis 10:00 Uhr via Ilias

Einführungsaufgabe: Gaussverteilung darstellen

Bevor wir mit der eigentlichen Aufgabe 4 beginnen, wollen wir uns zunächst dem "Würfeln" (= Erzeugen einer Verteilung durch die Generierung von Zufallszahlen) und Darstellen der Gaussverteilung widmen.

In [33]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
```

Dazu erstellen wir zunächst eine Funktion `gauss` mit der wir die Gaussfunktion

$$f(x) = \frac{1}{\sqrt{2}} \exp\left(\frac{1}{2} \frac{(x \quad \mu)^2}{2} \right)$$

implementieren können.

In [34]:
```
def gauss(x, n, sig, mu):
    return n/(np.sqrt(2*np.pi)*sig) * np.exp(-np.power(x - mu, 2.) / (2 * np.power(sig, 2.)))
```

Nun wollen wir Zufallszahlen generieren, die einer Verteilung genügen die dieser Funktion entspricht. Mit `numpy` ist das ganz einfach, denn diese Bibliothek enthält bereits eine Methode dafür, und zwar `normal` im Modul `random`. Wir generieren uns also eine Liste `x` mit 10000 normal-verteilte Zufallszahlen mit $\mu = 5$ und $= 1$:

In [35]:
```
x = np.random.normal(5,1,10000)
x
```

Out[35]:
```
array([6.25688727, 5.50645164, 4.76706611, ..., 6.19606489, 4.09142988,
       3.21419124])
```

Wir möchten nun sehen, wie gut (oder schlecht?) unsere Verteilung der obigen Funktion genügt. Das erkennt man üblicherweise dann am besten, wenn man es grafisch darstellt. Wir möchten also ein Diagramm erstellen mit der gewürfelten Verteilung als Histogramm und darüberliegend den Verlauf der tatsächlichen Gauss-Funktion. Wir wählen dazu das Intervall $x \quad [0, 10]$.

Zunächst brauchen wir eine Methode mit der wir die Liste der Bingrenzen definieren können in Abhängigkeit der Intervallgrenzen und der Anzahl gleichmäßig breiter Bins:

In [36]:
```
def listOfBins(xmin, xmax, nbins):
    return [xmin+(xmax-xmin)/nbins*i for i in range(nbins+1)] ## +1 since we need the upper bound as
```

In [37]:
```
listOfBins(0,10,10)
```

Out[37]:
```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0]
```

In [38]:
```
listOfBins(2,12,10)
```

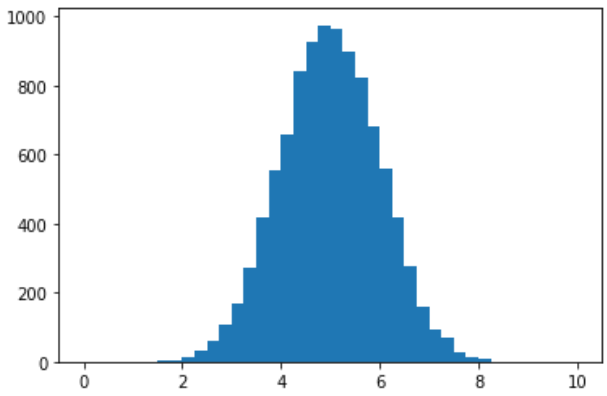
Out[38]:
```
[2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0]
```

In [39]:
```
listOfBins(2,12,40)
```

Out[39]:
```
[2.0,
 2.25,
 2.5,
 2.75,
 3.0,
 3.25,
 3.5,
 3.75,
 4.0,
 4.25,
 4.5,
 4.75,
 5.0,
 5.25,
 5.5,
 5.75,
 6.0,
 6.25,
 6.5,
 6.75,
 7.0,
 7.25,
 7.5,
 7.75,
 8.0,
 8.25,
 8.5,
 8.75,
 9.0,
 9.25,
 9.5,
 9.75,
 10.0,
 10.25,
 10.5,
 10.75,
 11.0,
 11.25,
 11.5,
 11.75,
 12.0]
```

Nun können wir das `pyplot` Histogramm erstellen, in dem wir die normal-verteilen Zufallswerte darstellen möchten. Das geht ganz einfach über die Funktion `hist`, der wir die Zufallszahlen und die Bingrenzen als Argumente übergeben. Wir verwenden 40 Bins von 0 bis 10. (Sie können mit der Anzahl der Bins herumexperimentieren um zu sehen, wie die Auflösung der Verteilung sich verändert.)

In [40]:
```
b = listOfBins(0,10,40)
plt.hist(x,bins=b)
plt.show()
```



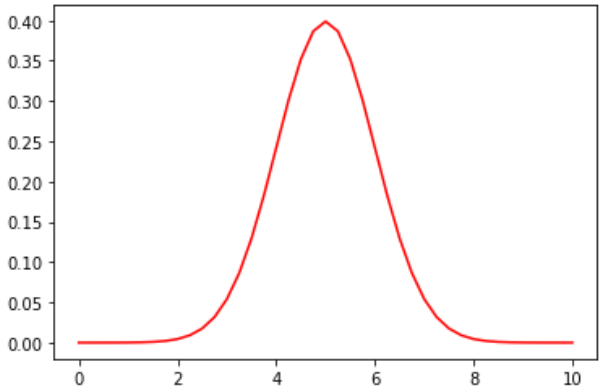
Nun möchten wir die Gaussfunktion $f(x)$ darüberlegen, um den Unterschied der zufallsgenerierten Verteilung zu feststellen zu können. Dazu verwenden wir die Funktion `plot` die mit x und $y = f(x)$ Werten gefüttert werden muss. Tatsächlich wird nicht eine Funktion gezeichnet sondern die Funktion wird an einzelnen Punkten (die wir als x Werte vorgeben) ausgewertet und die Punkte dann miteinander verbunden. Je mehr Punkte wir also übergeben, desto genauer und schöner wir die Kurve (wobei ab einer gewissen Anzahl Punkte keine Verbesserung mehr bei zusätzlichen Punkten feststellbar ist). Wir wählen 40 Punkte und lassen uns durch `numpy` 's Funktion `linspace` eine Liste von gleichmäßigen x Positionen generieren (auch wieder im Intervall $[0, 10]$):

In [41]:
```
xpos = np.linspace(0,10,40+1)
xpos
```

Out[41]:
```
array([ 0.  ,  0.25,  0.5 ,  0.75,  1.  ,  1.25,  1.5 ,  1.75,  2.  ,
        2.25,  2.5 ,  2.75,  3.  ,  3.25,  3.5 ,  3.75,  4.  ,  4.25,
        4.5 ,  4.75,  5.  ,  5.25,  5.5 ,  5.75,  6.  ,  6.25,  6.5 ,
        6.75,  7.  ,  7.25,  7.5 ,  7.75,  8.  ,  8.25,  8.5 ,  8.75,
        9.  ,  9.25,  9.5 ,  9.75, 10.  ])
```

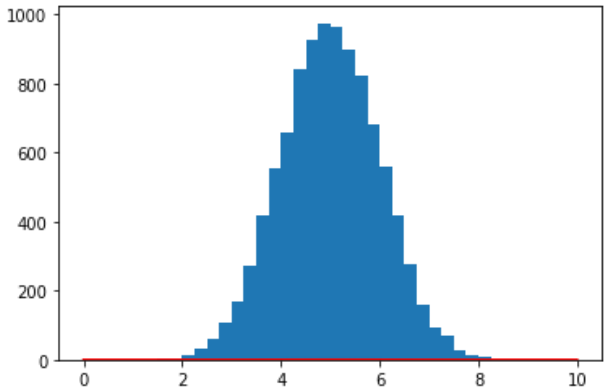
Nun werten wir die Funktion an diesen Punkten aus und stellen $f(x)$ dar:

In [42]:
```
plt.plot(xpos, gauss(xpos,1,1,5), color="red")
plt.show()
```



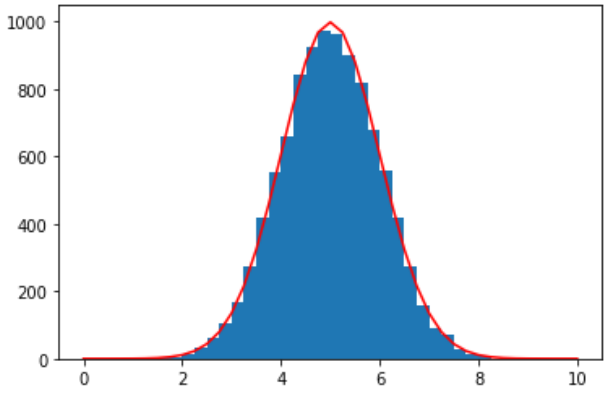
Die beiden Diagramme sollen nun allerdings übereinander dargestellt werden. Das erreichen wir, indem wir die beiden Befehle in einer gemeinsamen Zelle nacheinander ausführen.

In [43]:
```
b = listOfBins(0,10,40)
plt.hist(x, bins=b)
plt.plot(xpos, gauss(xpos, 1, 1, 5), color="red")
plt.show()
```



Was ist hier passiert? Die Kurve für $f(x)$ ist nicht mehr sichtbar! Das liegt daran, dass die Normierung der beiden Diagramme gegeneinander nicht passt. Gehen Sie nochmal zurück zu den Einzeldiagrammen und achten Sie auf die Werte auf der y Achse. Wir müssen also eines der Diagramme anpassen. Entweder das Histogramm indem wir die Einträge jedes Bins normieren, sodass das Integral (die Summe aller Einträge) gleich 1 ist. Oder wir skalieren die Darstellung von $f(x)$ nach oben, sodass das Integral von $f(x)$ dem Integral des Histogramms entspricht. Wir entscheiden uns für letzteres, da die Funktion `gauss` bereits einen Parameter `n` zur Skalierung enthält. Den setzen wir auf das Maximum des Histogramms. Dazu lassen wir uns die Binhöhen des Histogramms in `bh` zurückgeben. Bemerke, dass die Funktion `hist` drei Rückgabewerte hat (siehe https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html), wobei wir die anderen beiden aber nicht brauchen. Daraus berechnen wir `n` als das Integral des Histogramms dividiert durch `4` (da wir 40 bins für das Intervall $x \in [0, 10]$ verwenden).

In [44]:
```
b = listOfBins(0,10,40)
bh,bx,pt = plt.hist(x, bins=b)
ymax = sum(bh)
plt.plot(xpos, gauss(xpos, ymax/4, 1, 5), color="red")
plt.show()
```

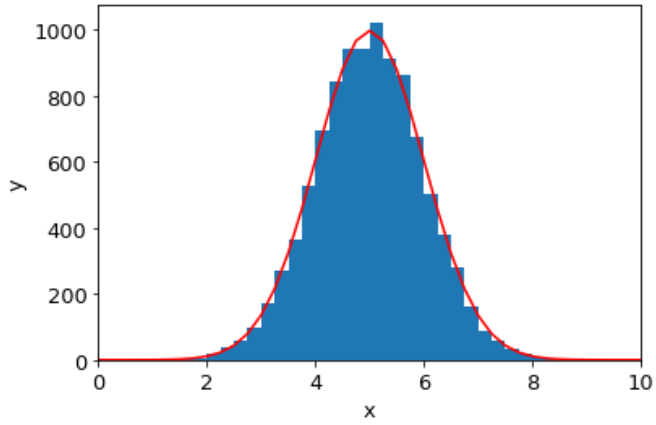


Die Abbildung sieht nun also genau so aus, wie wir es erwarten würden. Allerdings fehlen noch entscheidende Dinge. Und zwar die Bezeichnungen! Ein Diagramm ohne Bezeichnung ist sinnlos, da wir nicht wissen, welche Werte wir herauslesen können. Wir fügen nun also Achsenbezeichnungen und Ticks hinzu, damit das Diagramm lesbar wird. Diese Labels sollten standardmäßig zu jedem Diagramm hinzugefügt werden!

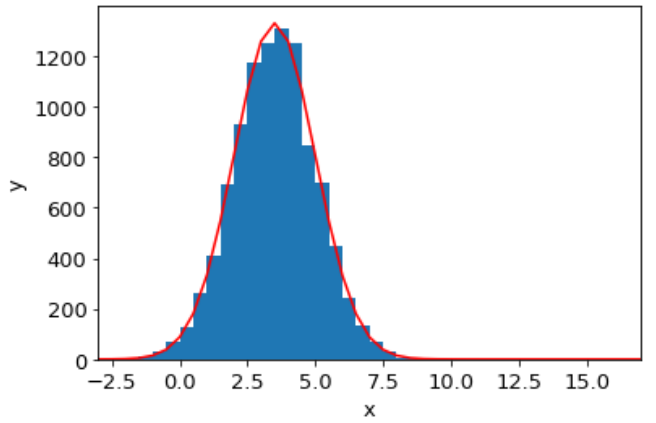
Außerdem bringen wir nun auch alles in einer Funktion zusammen, sodass wir einfach verschiedene Werte ausprobieren können.

In [45]:
```
def plotGauss(xmin, xmax, nbins, mu, sig):
    x = np.random.normal(mu, sig, 10000)
    b = listOfBins(xmin, xmax, nbins)
    bh,bx,pt = plt.hist(x, bins=b)
    ymax = sum(bh)*(xmax-xmin)/nbins
    xpos = np.linspace(xmin, xmax, nbins+1)
    plt.plot(xpos, gauss(xpos, ymax, sig, mu), color="red")
    plt.xlabel("x", fontsize=13)
    plt.ylabel("y", fontsize=13)
    plt.xlim(xmin, xmax)
    plt.xticks(fontsize=13)
    plt.yticks(fontsize=13)
    plt.show()
```

In [46]:
```
plotGauss(0, 10, 40, 5, 1)
```



In [47]:
```
plotGauss(-3, 17, 40, 3.5, 1.5)
```



Aufgabe 3: Binomial- und Poissonverteilungen (8P) 3/8

In dieser Aufgabe wollen wir testen, ob die Binomial- und Poissonverteilungen sich für große n bzw. tatsächlich an die Gaussverteilung annähern.

a) Binomialverteilung (6P)

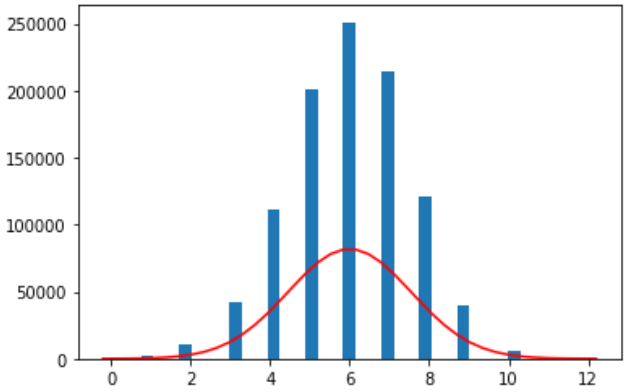
Befüllen Sie ein Histogramm mit 1M nach der Binomialverteilung verteilten diskreten Zufallsvariablen x für p = 0.6 und der Anzahl von Versuchen n (s. Vorlesung). Um den Wertebereich von x festzulegen der gezeigt werden soll, berechnen Sie mit Hilfe der Standardabweichung der Binomialverteilung die Grenzen des 4 Bereiches um den Mittelwert der Verteilung. Zeichnen Sie die Gaussfunktion mit geeigneten Werten von μ und darüber. Geben Sie außerdem den Unterschied zwischen den beiden Verteilungen an indem Sie die Differenzen (Achtung: jeweils die Absolutbeträge der Differenzen!) der Binhöhe und der Werte der Gaussfunktion an jedem Binmittelpunkt aufsummieren und durch das Integral der Gaussfunktion teilen (also Sie berechnen die relative Abweichung der Verteilung im Histogramm in Bezug auf die Normalverteilung). Testen Sie verschiedene Werte für die Anzahl der Versuche n. Wie groß muss n sein, damit die Abweichung kleiner als 20% ist?

In [48]:
```
def plots(n):
    x = np.random.binomial(n, 0.6, 1000000)
    mu = n * 0.6
    sigma = np.sqrt(n * 0.6 * (1-0.6))
    a = listOfBins(-4*sigma+mu, 4*sigma+mu, 39)
    bhe, bx, bt = plt.hist(x, bins=a)
    ymaxx = sum(bhe) * (binsmax-binmin)/nbins
    xss = np.linspace(-4 * sigma + mu, 4 * sigma + mu, 39 + 1)

    normal = gauss(np.linspace(-4 * sigma - mu, 4 * sigma + mu, 39),
                   ymaxx / (39 / (4 * sigma + 4 * sigma)), sigma, mu)
    abweichung = sum(bhe-normal) / sum(xss) * 100
    print("Relative Abweichung für n={0:1} ist {1:1.5f} Prozent".format(
        n, abweichung))
    plt.plot(xss, gauss(xss, ymaxx / (39 / (4 * sigma + 4 * sigma)),
             sigma, mu), color="red")
    plt.show()

plots(10)
plots(5349)
plots(5350)
plots(10000)
```

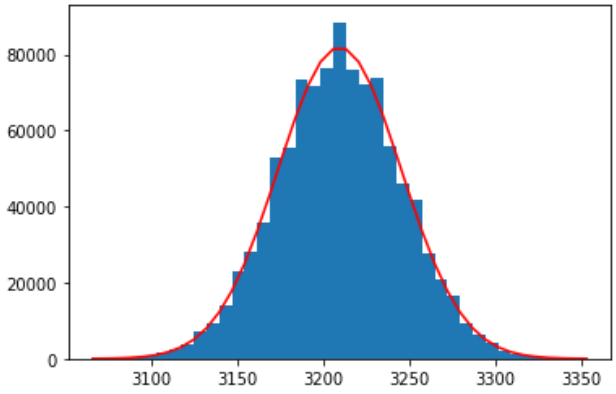
die Anzahl der bin konstant zu machen ist unpraktisch bei verschiedenen n

um die Abweichung zu Berechnen muss man die gauss funktion in der binmitte auswerten außerdem muss der Betrag verwendet werden

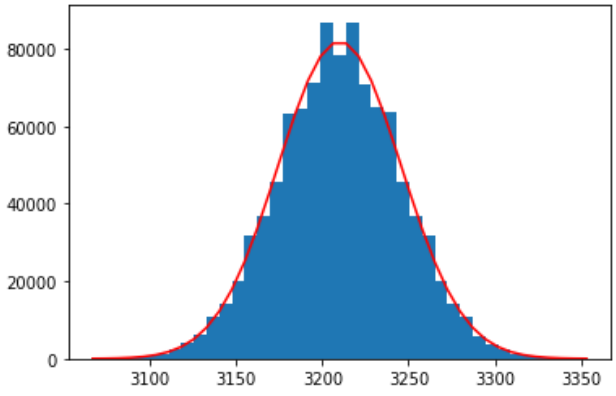
```
Relative Abweichung für n=10 ist 210402.64244 Prozent
```




```
Relative Abweichung für n=5349 ist 737.33757 Prozent
```



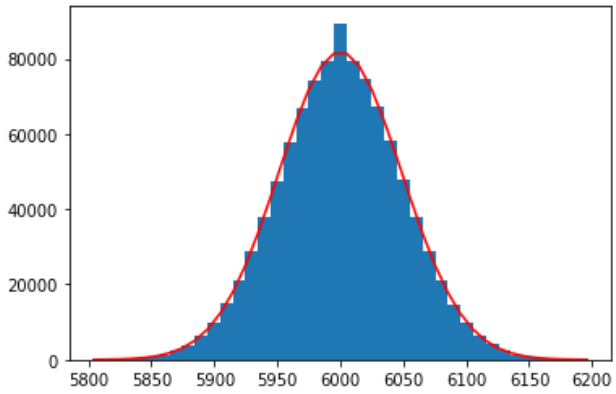

```
Relative Abweichung für n=5350 ist 737.22406 Prozent
```




```
Relative Abweichung für n=10000 ist 415.63094 Prozent
```

+ Achsenbeschriftungen

2/6



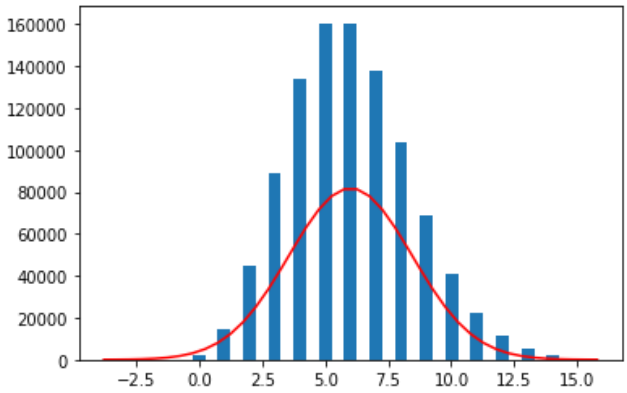
Antwort: n muss grösser als 5349 sein.

b) Poissonverteilung (2P)

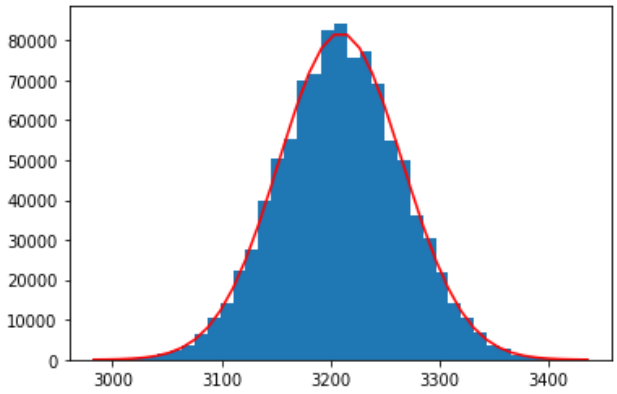
Wiederholen Sie Aufgabenteil (a) für die Poissonverteilung. Wiederum die Frage: wie groß muss sein damit der relative Fehler kleiner als 20% ist?

In [49]:
```
def plots1(n):
    lmd = 0.6 * n
    xs = np.random.poisson(lmd, 1000000)
    std = np.sqrt(lmd)
    d = listOfBins(-4 * std + lmd, 4 * std + lmd, 39)
    h, f, t = plt.hist(xs, bins=d)
    maxx = sum(h)
    s = np.linspace(-4 * std + lmd, 4 * std + lmd, 40)

    normal1 = gauss(np.linspace(-4 * std + lmd, 4 * std + lmd, 39),
                    maxx / (39 / (4 * std + 4 * std)), std, lmd)
    abweichung = sum(h - normal1) / sum(s) * 100
    print("Relative Abweichung für n={0:1} ist {1:1.5f} Prozent".format(
        n, abweichung))

    plt.plot(s, gauss(s, maxx/(39/(4*std+4*std)), std, lmd), color="red")
    plt.show()


plots1(10)
plots1(5349)
plots1(5350)
plots1(10000)
```

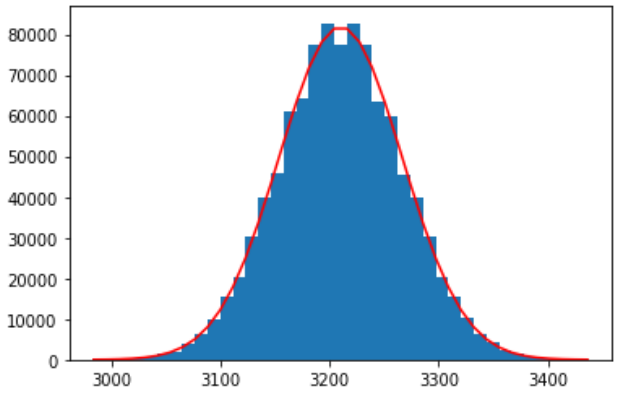
```
Relative Abweichung für n=10 ist 10694.38547 Prozent
```



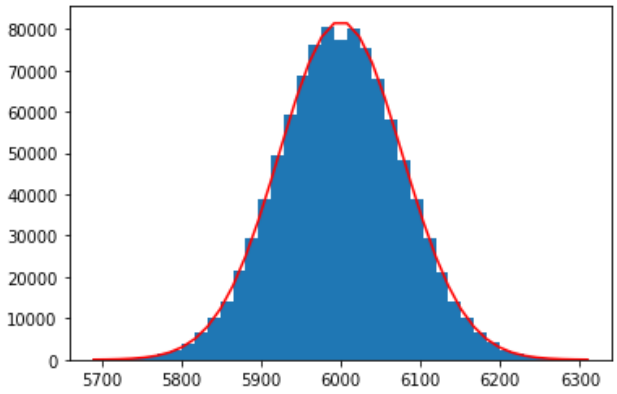

```
Relative Abweichung für n=5349 ist 20.00196 Prozent
```




```
Relative Abweichung für n=5350 ist 19.99811 Prozent
```




```
Relative Abweichung für n=10000 ist 10.69890 Prozent
```



Antwort: n muss auch grösser als 5349 sein.

Aufgabe 4: Zentraler Grenzwertsatz (8P)

7/8 wegen fehlenden Achsenbeschriftungen

Der Zentrale Grenzwertsatz besagt, dass die Summe von ausreichend vielen unabhängigen Zufallsvariablen aus der gleichen (aber beliebigen!) Wahrscheinlichkeitsdichtefunktion durch eine Gaussverteilung beschrieben werden kann. Um den zentralen Grenzwertsatz zu demonstrieren, sollen in dieser Übung mehrere im Intervall $[0, 1]$ gleichverteilte Zufallsvariablen addiert werden, und mit einer Gausschen Wahrscheinlichkeitsdichtefunktion verglichen werden.

a) Zufallszahlen (1P)

1/1

Schreiben Sie zunächst eine Funktion, mit der Sie `NVar` Zufallszahlen im Intervall $[0, 1]$ via der `numpy` Funktion `random.uniform` gezogen werden, und die Summe von diesen zurückgegeben wird.

In [50]:
```python
def sums(NVar):
    numbers = np.random.uniform(0, 1, NVar)
    return sum(numbers)
```

✓

b) Grenzwertsatz (5P)

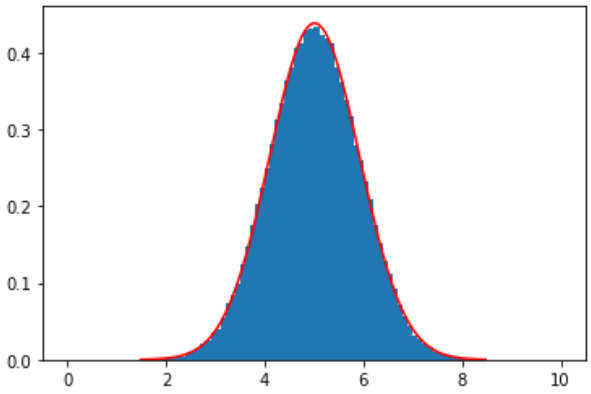
In einer weiteren Funktion `grenzwertsatz` sollen n Zufallszahlen, die die Summe von jeweils `NVar` gleichverteilten Zufallszahlen (Ihre Funktion in Teil (a)!) sind, in ein Histogramm eingetragen werden. Da nun ein Vergleich mit einer Gaussverteilung gemacht werden soll, soll diese neue Funktion zusätzlich die Gaussverteilung im Diagramm oberhalb des Histogramms anzeigen.

Bemerkung: Hier machen wir uns jetzt das Leben leichter, indem wir das Histogram auf 1 normieren, und damit auch die Normierung der Gaussverteilung nicht mehr raten müssen. Das geht ganz einfach in dem man in der `hist` Funktion am Ende `density=True` einfügt. Für den Mittelwert und die Standardabweichung der Gaussverteilung die zum Vergleich verwendet wird koennen Sie einfach den Mittelwert und Standardabweichung des Histogramms ausrechnen.

In [51]:
```
def grenzwertsatz(n, NVar, name):
    nbins = NVar * 10
    bins = [NVar*i/nbins for i in range(nbins+1)]
    x = []
    for i in range(n):
        x.append(sums(NVar))
        i += 1
    xx = np.linspace(min(x), max(x), n)
    plt.plot(xx, gauss(xx, 1, np.std(x), np.mean(x)), color="red")
    plt.hist(x, bins, density=True)
    plt.show()

grenzwertsatz(100000, 10, name=None)
```

das geht einfacher mit numpy

z.B. so
def sum_zufallszahl_array(NVar:int, n:int) -> np.ndarray:
    zufallszahl_array = np.random.uniform(0, 1, size=(NVar, n))
    return np.sum(zufallszahl_array, axis=0)

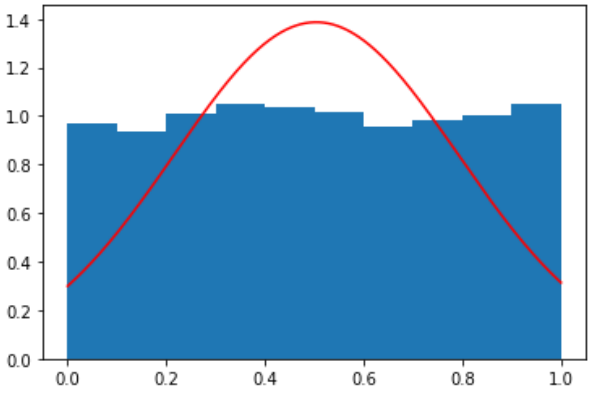


+ Achsenbeschriftungen

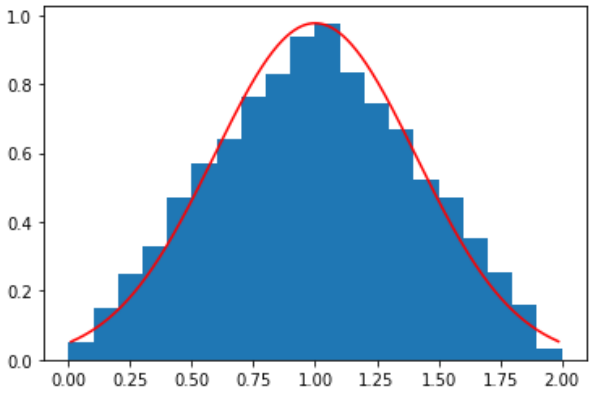
Starten Sie zunächst mit nur einer Zufallsvariablen (`NVar=1`). Erhöhen Sie nun die Anzahl schrittweise. Wieviele gleichverteilte Zufallsvariablen sollte man mindestens addieren, um eine gute Annäherung an die Gaussfunktion zu erhalten?

In [52]:
```
for i in range(1, 15):
    print("Nvar", i)
    grenzwertsatz(10000, i, name=None)
```

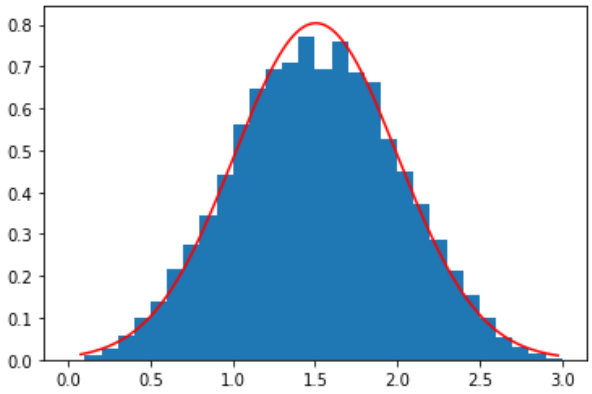
```
Nvar 1
```



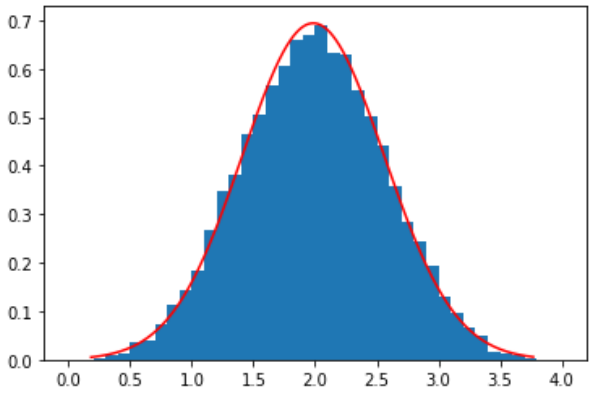

```
Nvar 2
```



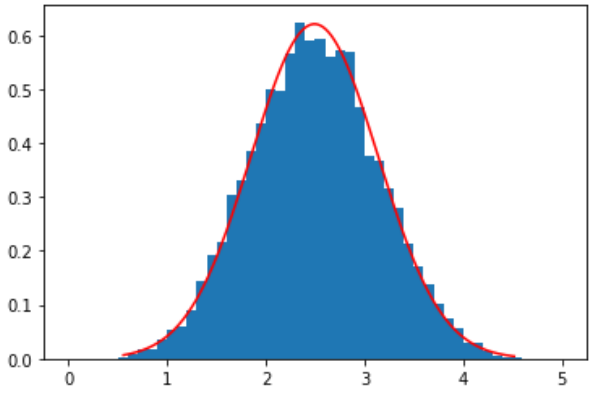
Nvar 3



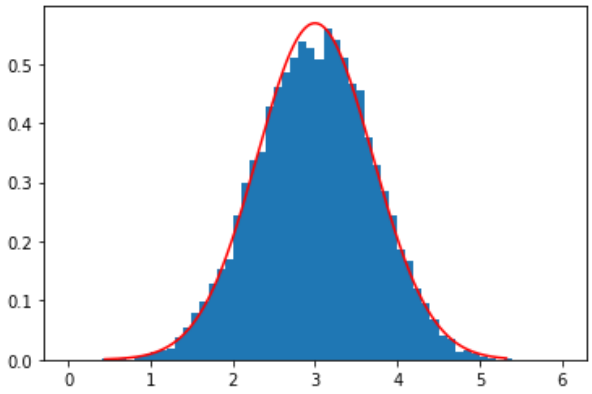
Nvar 4



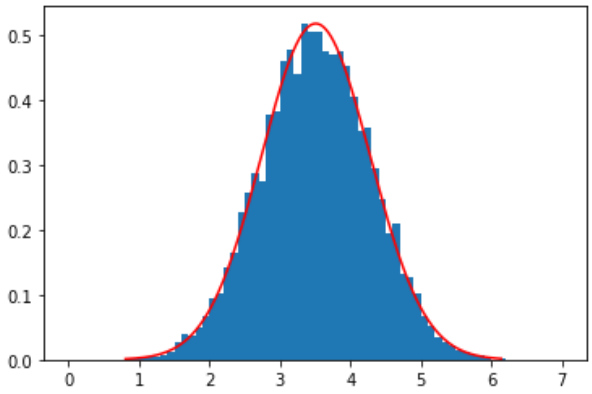
Nvar 5



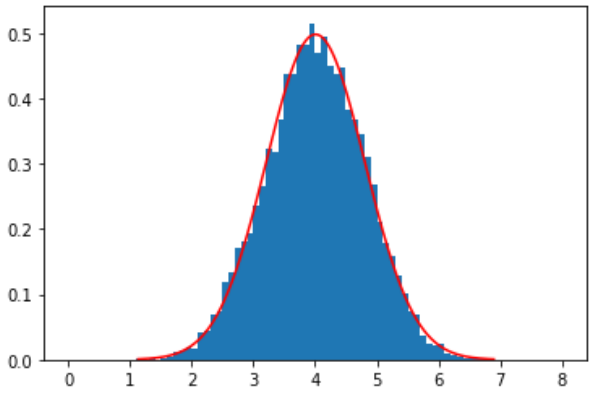
Nvar 6



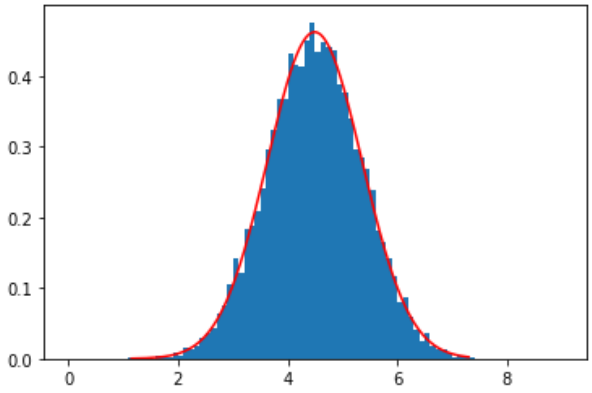
Nvar 7



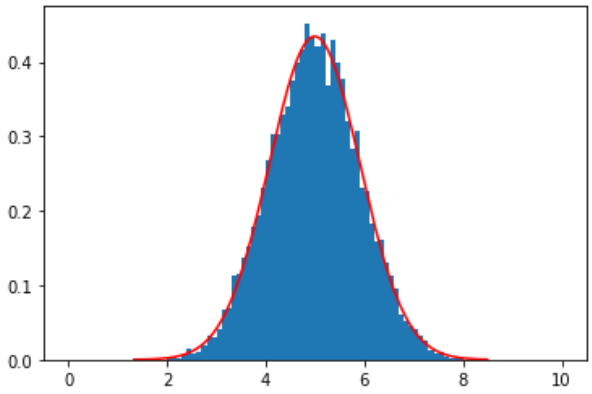
Nvar 8



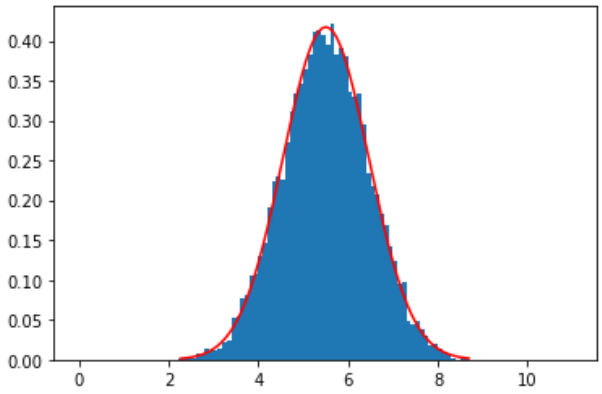
Nvar 9



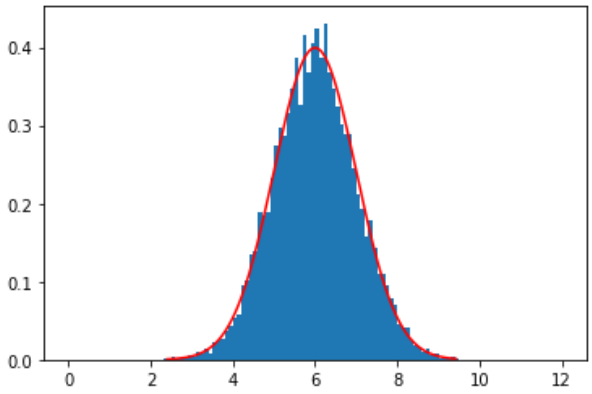
Nvar 10



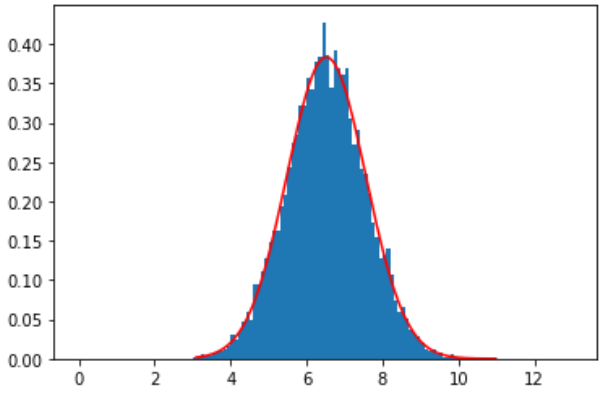
Nvar 11



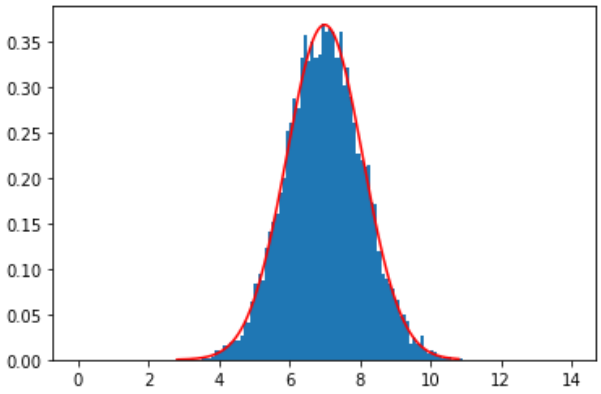
Nvar 12



Nvar 13



Nvar 14



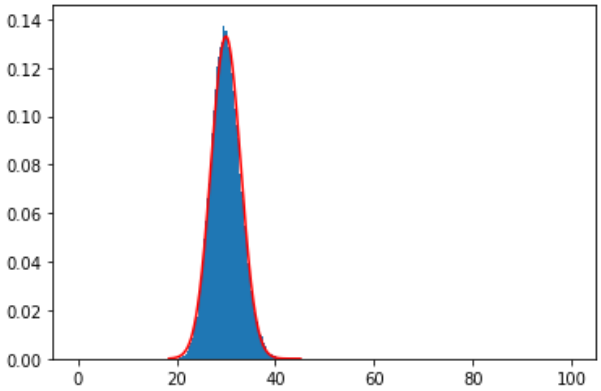
Annäherung an die Gaussfunktion

Aus meiner Perspektive braucht es mindestens 7 Zufallsvariablen zu addieren, um eine gute Annäherung an die Gaussfunktion zu erhalten. .

c) Exponentialverteilung (2P)

Adaptieren Sie Ihre Funktionen aus Teil (a) und (b) insofern, dass Sie nun exponentiell-verteilte Zufallsvariablen generieren (mit der Zerfallskonstante $\mu = 0.3$) und den Grenzwertsatz ein weiteres mal verifizieren. Wie unterscheidet sich die benoetigte Stichprobengroesse fuer die Gleichverteilung und die Exponentialverteilung?

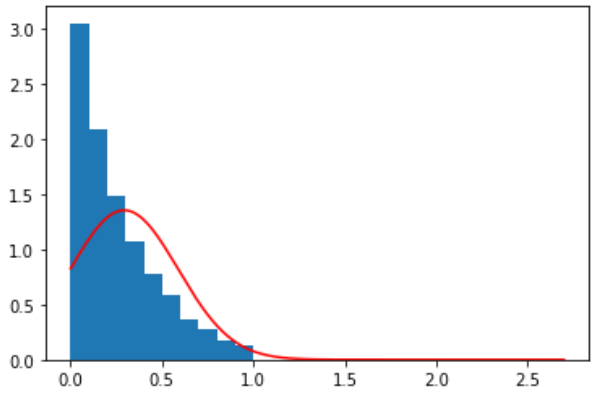
In [53]:
```
def sums2(NVar):
    numbers = np.random.exponential(0.3, NVar)
    return sum(numbers)


def grenzwertsatz2(n, NVar, name):
    nbins = NVar * 10
    bins = [NVar*i/nbins for i in range(nbins+1)]
    x = []
    for i in range(n):
        x.append(sums2(NVar))
        i += 1
    xx = np.linspace(min(x), max(x), n)
    plt.plot(xx, gauss(xx, 1, np.std(x), np.mean(x)), color="red")
    plt.hist(x, bins, density=True)
    plt.show()

grenzwertsatz2(100000, 100, name=None)
```



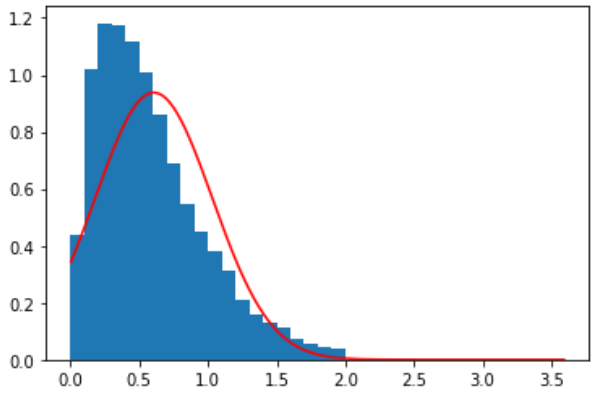
Bei niedrigen NVar-Werten ist die Verteilung schief, sodass eine höhere Anzahl von Zufallsvariablen benötigt wird, um eine Normalverteilung zu bekommen. Siehe unten.

In [54]:
```
for i in range(1,15):
    print("Nvar=",i)
    grenzwertsatz2(10000, i, name=None)
```

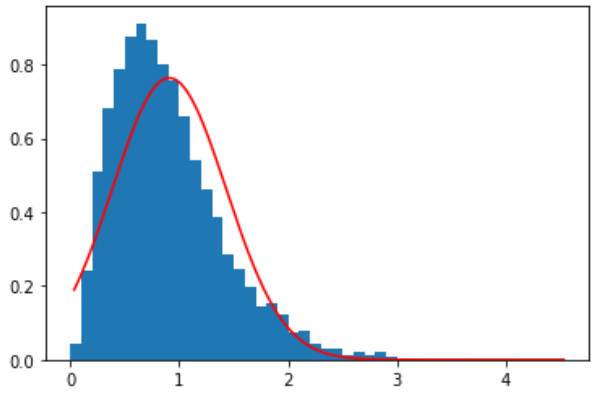
```
Nvar= 1
```



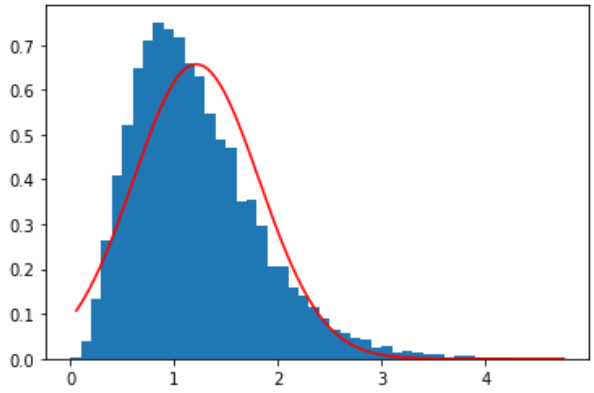
Nvar= 2



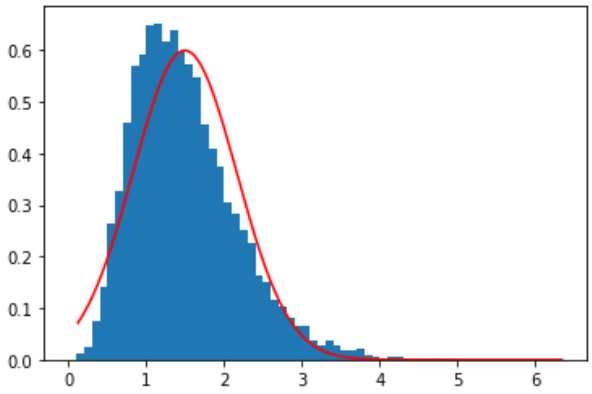
Nvar= 3



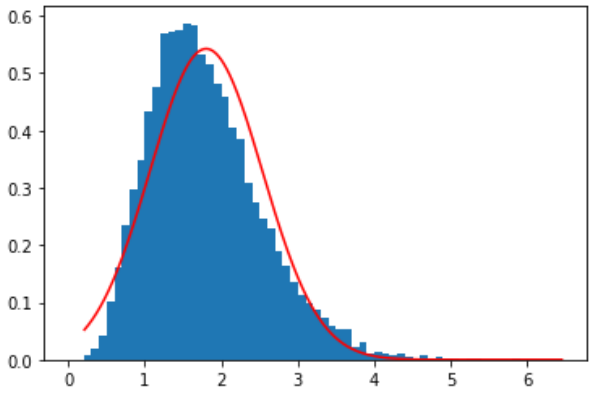
Nvar= 4



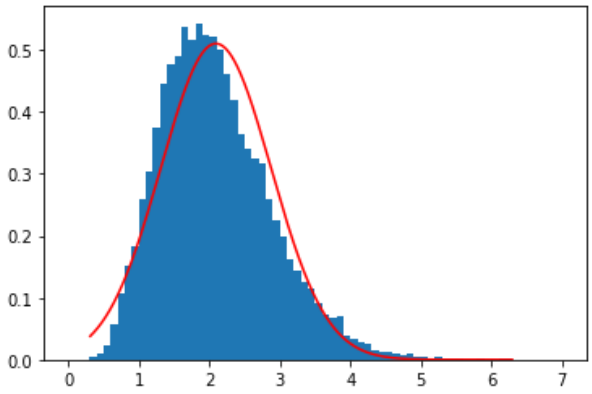
Nvar= 5



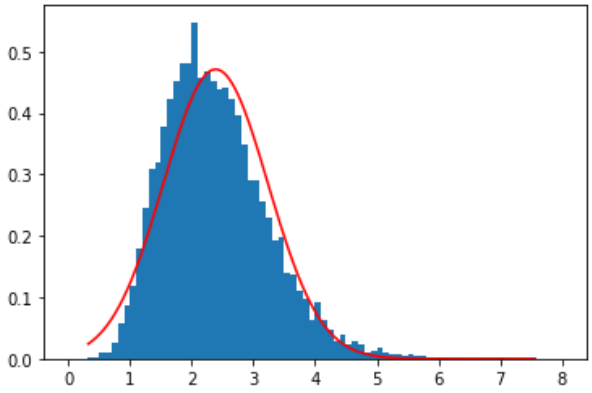
Nvar= 6



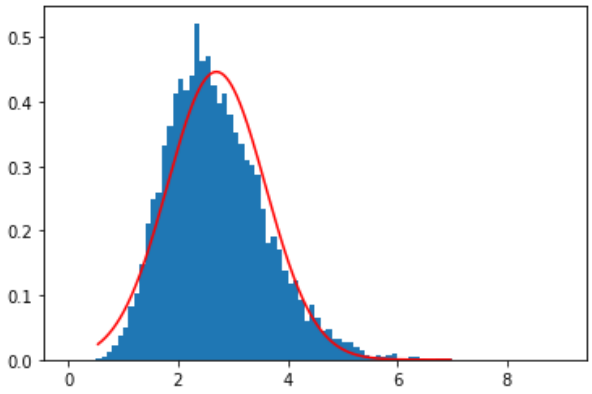
Nvar= 7



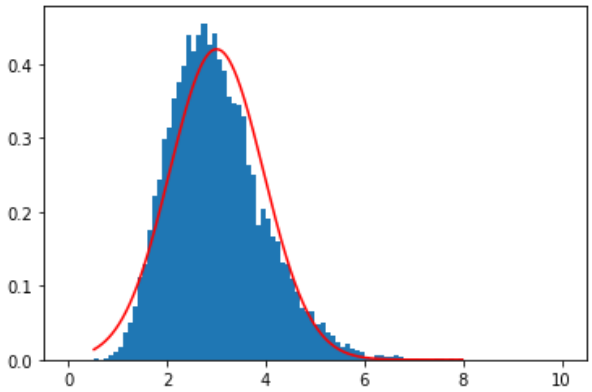
Nvar= 8



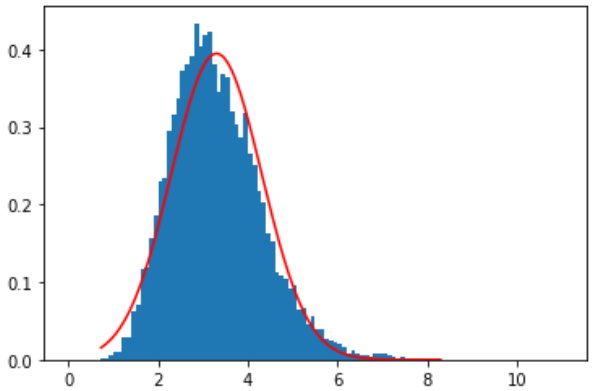
Nvar= 9



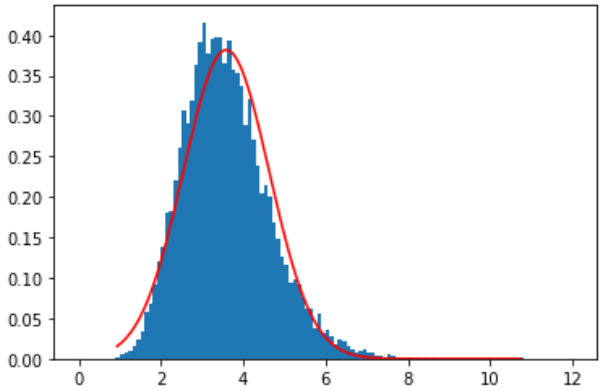
Nvar= 10



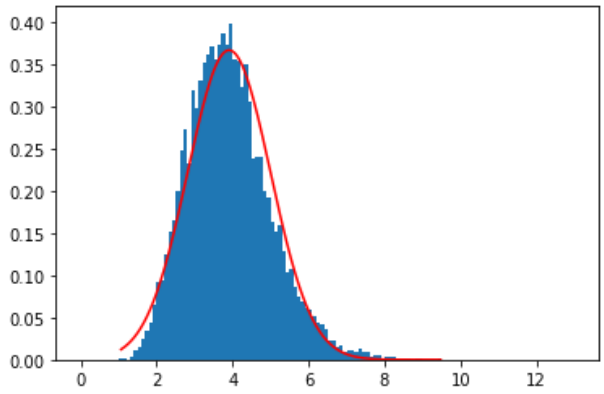
Nvar= 11



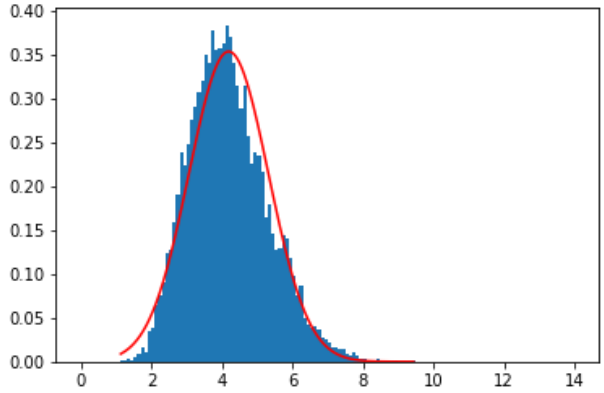
Nvar= 12



Nvar= 13


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
12

Nvar= 14


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8
10
12
14