Chapter 3: statistics of continuous variables

Univariate probability distribution and normal distribution

Let us start with a random variable $x \in R$ following the probability density function (PDF) $p : R \to R$ with

$$\int dx p(x) = 1 \,. \tag{1}$$

Since we only consider one variable, we call such distributions **univariate**.

We define the expectation value of a function $f : R \to C$ of this variable as

$$\langle f(x) \rangle = \int dx p(x) f(x) \,. \tag{2}$$

Next, we call $\langle x^m \rangle$ the m-th **moment** of the distribution with first non-trivial case

$$\mu = \langle x \rangle$$

the **mean** of the distribution. We refer to $\langle (x-\mu)^m \rangle$ as the m-th **central moment** of the distribution and the first non-trivial case

$$\sigma^2 = \langle (x-\mu)^2 \rangle$$

as the **variance** of the distribution and call σ the **standard deviation**.

We define the fraction of results that fall within $n\,\sigma$ around the mean as

$$f_n = \int_{\mu-n\sigma}^{\mu+n\sigma} dx p(x) \tag{3}$$

which depends on the PDF. A particularly important case is the **normal distribution** with

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

for which the PDF only depends on the mean and the variance. For the normal distribution, the f_n are independent of μ and σ (***Homework***: show this independence) and

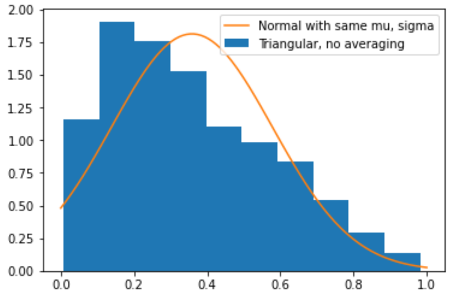
$$f_1 = 0.682\ldots \,, \tag{5}$$
$$f_2 = 0.954\ldots \,, \tag{6}$$
$$f_3 = 1 - 2.6\ldots \times 10^{-3} \,, \tag{7}$$
$$f_4 = 1 - 6.3\ldots \times 10^{-5} \,, \tag{8}$$
$$f_5 = 1 - 5.7\ldots \times 10^{-7} \,. \tag{9}$$

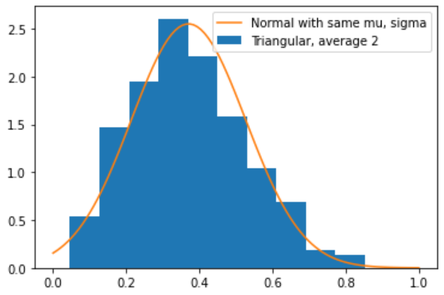
So if we know the PDF, we can translate fluctuations of size $n\,\sigma$ to probabilities of such a fluctuation. And if the PDF is normal, the f_n are fixed numbers.

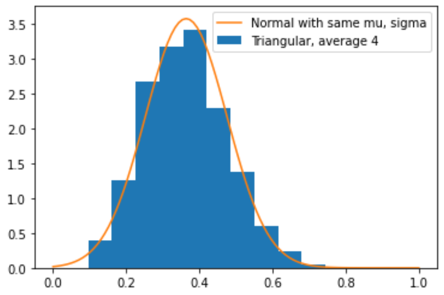
Interestingly, the normal distribution naturally appears in averaging processes. Let us first look at some data:

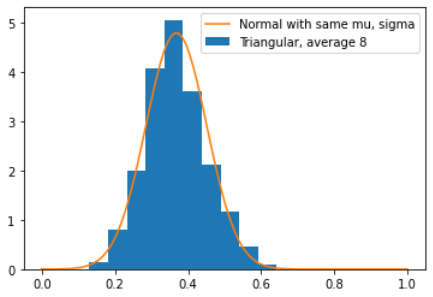
In [1]:
```python
import random
import numpy as np
from matplotlib import pyplot as plt

random.seed(13)
```

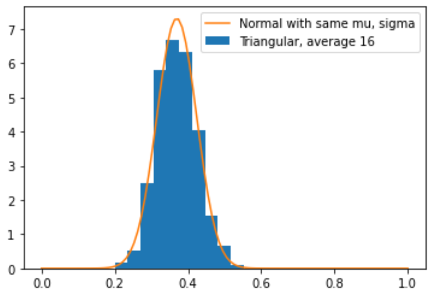
In [2]:
```python
def histogram(X, l):
    v = [X() for i in range(1000)]
    plt.hist(v,label=l,density=True)
    mu = np.mean(v)
    sigma = np.std(v)
    x = np.linspace(0, 1.0, 100)
    y = [(2.*np.pi*sigma**2.)**-0.5*np.exp(-(xv-mu)**2./2/sigma**2.) for xv in x]
    plt.plot(x, y, label="Normal with same mu, sigma")
    plt.legend()
    plt.show()

# start with triangular distribution
def p():
    return random.triangular(0.0,1.0,0.1)

histogram(p, "Triangular, no averaging")
histogram(lambda: sum([p() for i in range(2)]) / 2.0, "Triangular, average 2")
histogram(lambda: sum([p() for i in range(4)]) / 4.0, "Triangular, average 4")
histogram(lambda: sum([p() for i in range(8)]) / 8.0, "Triangular, average 8")
histogram(lambda: sum([p() for i in range(16)]) / 16.0, "Triangular, average 16")
```











Homework: Repeat this exercise with different starting distributions.

Central limit theorem

The effect we just observed can be shown in a straightforward way.

Consider n random variables $x_1, \dots, x_n$ drawn from the identical PDF $p(x)$. They should be independent, i.e., their joint probability factors

$$p(x_1, \dots, x_n) = p(x_1) \cdots p(x_n) \, . \tag{10}$$

We then define $\tilde{x}$ as the sum of these n random variables

$$\tilde{x} = \sum_{i=1}^{n} x_i \, . \tag{11}$$

We first note that

$$\tilde{\mu} = \langle \tilde{x} \rangle = \langle \sum_{i=1}^{n} x_i \rangle = \int dx_1 \cdots dx_n p(x_1, \ldots, x_n) \sum_{i=1}^{n} x_i \tag{12}$$

$$= \sum_{i=1}^{n} \int dx_1 \cdots dx_n p(x_1) \cdots p(x_n) x_i \tag{13}$$

$$= \sum_{i=1}^{n} \int dx_i p(x_i) x_i \tag{14}$$

$$= \sum_{i=1}^{n} \mu = n\mu \tag{15}$$

and

$$\tilde{\sigma}^2 = \langle \tilde{x}^2 \rangle - \tilde{\mu}^2 \tag{16}$$

$$= \sum_{i,j=1}^{n} \langle x_i x_j \rangle - n^2 \mu^2 \tag{17}$$

$$= \sum_{i=1}^{n} \langle x^2 \rangle + \sum_{i \neq j} \langle x \rangle^2 - n^2 \mu^2 \tag{18}$$

$$= n(\sigma^2 + \mu^2) + n(n-1)\mu^2 - n^2 \mu^2 \tag{19}$$

$$= n\sigma^2 \, . \tag{20}$$

It is then natural to define the helper variable

$$z = \frac{\tilde{x} - n\mu}{\sqrt{n\sigma^2}} \tag{21}$$

which has mean 0 and variance 1.

We then define the **characteristic function** of the PDF as its Fourier transform

$$\phi(k) = \int dx p(x) e^{ikx} \, , \tag{22}$$

$$p(x) = \frac{1}{2\pi} \int dk \phi(k) e^{-ikx} \, . \tag{23}$$

The PDF for z is then

$$p_z(z) = \int dx_1 \cdots dx_n \delta\left(z - \frac{\sum_{i=1}^{n} x_i - n\mu}{\sqrt{n}\sigma}\right) p(x_1, \ldots, x_n) \, , \tag{24}$$

$$= \int dx_1 \cdots dx_n \delta\left(z - \frac{\sum_{i=1}^{n} x_i - n\mu}{\sqrt{n}\sigma}\right) p(x_1) \cdots p(x_n) \, , \tag{25}$$

$$= (2\pi)^{-n} \int dx_1 \cdots dx_n dk_1 \cdots dk_n \delta\left(z - \frac{\sum_{i=1}^{n} x_i - n\mu}{\sqrt{n}\sigma}\right) e^{-i\sum_{i=1}^{n} x_i k_i} \phi(k_1) \cdots \phi(k_n) \, , \tag{26}$$

and therefore the characteristic function for z is

$$\phi_z(\omega) = (2\pi)^{-n} \int dz \int dx_1 \cdots dx_n dk_1 \cdots dk_n \delta\left(z - \frac{\sum_{i=1}^{n} x_i - n\mu}{\sqrt{n}\sigma}\right) e^{-i\sum_{i=1}^{n} x_i k_i + i\omega z} \phi(k_1) \cdots \phi(k_n) \, , \tag{27}$$

$$= (2\pi)^{-n} \int dx_1 \cdots dx_n dk_1 \cdots dk_n e^{-i\sum_{i=1}^{n} x_i (k_i - \frac{\omega}{\sqrt{n}\sigma}) - i\omega\sqrt{n}\frac{\mu}{\sigma}} \phi(k_1) \cdots \phi(k_n) \, , \tag{28}$$

$$= \int dk_1 \cdots dk_n \left[\prod_{i=1}^{n} \delta\left(k_i - \frac{\omega}{\sqrt{n}\sigma}\right)\right] e^{-i\omega\sqrt{n}\frac{\mu}{\sigma}} \phi(k_1) \cdots \phi(k_n) \, , \tag{29}$$

$$= e^{-i\omega\sqrt{n}\frac{\mu}{\sigma}} \phi\left(\frac{\omega}{\sqrt{n}\sigma}\right)^n \, , \tag{30}$$

$$= e^{-i\omega\sqrt{n}\frac{\mu}{\sigma}} \left(\phi(0) + \phi'(0)\frac{\omega}{\sqrt{n}\sigma} + \frac{1}{2}\phi''(0)\frac{\omega^2}{n\sigma^2} + O(n^{-3/2})\right)^n \, . \tag{31}$$

Since

$$\phi(0) = \int dx p(x) = 1 \, , \tag{32}$$

$$\phi'(0) = i \int dx x p(x) = i\mu \, , \tag{33}$$

$$\phi''(0) = -\int dx x^2 p(x) = -(\sigma^2 + \mu^2) \, , \tag{34}$$

and

$$\log(1 + \varepsilon a_1 + \varepsilon^2 a_2 + O(\varepsilon^3)) = a_1 \varepsilon + (a_2 - \frac{1}{2} a_1^2) \varepsilon^2 + O(\varepsilon^3) \tag{35}$$

we have

$$\phi_z(\omega) = e^{-i\omega\sqrt{n}\frac{\mu}{\sigma}}\left(1 + \phi'(0)\frac{\omega}{\sqrt{n}\sigma} + \frac{1}{2}\phi''(0)\frac{\omega^2}{n\sigma^2} + O\left(\frac{1}{n^{3/2}}\right)\right)^n \tag{36}$$

$$= e^{-i\omega\sqrt{n}\frac{\mu}{\sigma}}e^{i\mu\sqrt{n}\frac{\omega}{\sigma}-\frac{1}{2}\sigma^2\frac{\omega^2}{\sigma^2}} + O(n^{-1/2}) \tag{37}$$

$$= e^{-\frac{1}{2}\omega^2} + O\left(\frac{1}{\sqrt{n}}\right) . \tag{38}$$

A simple Gaussian integral then yields

$$p_z(z) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^2} + O\left(\frac{1}{\sqrt{n}}\right) . \tag{39}$$

We have just proved the **central limit theorem** that states that the sum or average of independent identically distributed random variables tends towards a normal distribution in the limit of many terms in the sum. In the case of an average, the mean of the average is the mean of the original random variables, while the variance is reduced by $1/n$.

Estimators and bias

Consider sampling n variables $x_1, \dots, x_n \in R$ from a PDF $p(x)$ and the samples to be independent of each other. Then we call

$$\hat{f}(x_1, \dots, x_n) \tag{40}$$

an **estimator** of a functional

$$f[p] \tag{41}$$

if

$$\lim_{n\to\infty} \hat{f}(x_1, \dots, x_n) = f[p] . \tag{42}$$

We call such an estimator **unbiased** if also

$$\langle \hat{f}(x_1, \dots, x_n)\rangle = \int dx_1 \cdots dx_n p(x_1, \dots, x_n)\hat{f}(x_1, \dots, x_n) = f[p] \tag{43}$$

and **biased** if this property does not hold. An estimator is therefore unbiased if repeatedly sampling a small number of variables n and averaging over the result of each gives the correct result.

Let us first consider a simple unbiased estimator for the mean of a distribution,

$$\hat{\mu}(x_1, \dots, x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{44}$$

for which we already showed above that

$$\langle \hat{\mu}(x_1, \dots, x_n)\rangle = \int dx_1 \cdots dx_n p(x_1, \dots, x_n)\hat{\mu}(x_1, \dots, x_n) \tag{45}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \int dx_1 \cdots dx_n p(x_1) \cdots p(x_n) x_i \tag{46}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \int dx_i p(x_i) x_i \tag{47}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \langle x\rangle = \langle x\rangle . \tag{48}$$

Interestingly, the estimator of the variance

$$\hat{\sigma}_b^2(x_1, \dots, x_n) = \frac{1}{n}\sum_{i=1}^{n} (x_i - \hat{\mu}(x_1, \dots, x_n))^2 \tag{49}$$

is biased, since

$$\langle \hat{\sigma}_b^2(x_1, \dots, x_n)\rangle = \frac{1}{n}\sum_{i=1}^{n} \langle x_i^2\rangle - \frac{1}{n^2}\sum_{i,j=1}^{n} \langle x_i x_j\rangle \tag{50}$$

$$= \left(\frac{1}{n} - \frac{1}{n^2}\right)\sum_{i=1}^{n} \langle x_i^2\rangle - \frac{1}{n^2}\sum_{i\neq j} \langle x_i\rangle\langle x_j\rangle \tag{51}$$

$$= \frac{n-1}{n}\langle x^2\rangle - \frac{n(n-1)}{n^2}\langle x\rangle^2 \tag{52}$$

$$= \frac{n-1}{n}(\langle x^2\rangle - \langle x\rangle^2) \tag{53}$$

which differs from the variance

$$\sigma^2 = \langle x^2\rangle - \langle x\rangle^2 . \tag{54}$$

The result, however, already tells us that an unbiased estimator can be defined by

$$\hat{\sigma}^2(x_1, \dots, x_n) = \frac{n}{n-1}\hat{\sigma}_b^2(x_1, \dots, x_n) \tag{55}$$

$$= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \hat{\mu}(x_1, \dots, x_n))^2 \,. \tag{56}$$

Let us demonstrate this numerically:

In [3]:
```
random.seed(13)

def estimate_variance_biased(x):
    mean = sum(x) / len(x)
    return sum([ (xi - mean) ** 2.0 for xi in x]) / len(x)

def estimate_variance_unbiased(x):
    mean = sum(x) / len(x)
    return sum([ (xi - mean) ** 2.0 for xi in x]) / (len(x) - 1)

def test_variance_estimator(est, n, N = 10000):
    sigma_expected = 1.0
    E = [est([random.gauss(mu=1.0, sigma=1.0)/sigma_expected**2. for i in range(n)]) for j in range(N)]
    val = np.mean(E)
    err = np.std(E) / N**0.5
    return f"{val} +- {err}"

for n in [4,8,16,32,64]:
    print(f"Biased estimator test for n={n}: 1 ==", test_variance_estimator(estimate_variance_biased, n))

print()

for n in [4,8,16,32,64]:
    print(f"Unbiased estimator test for n={n}: 1 ==", test_variance_estimator(estimate_variance_unbiased, n))
```

```
Biased estimator test for n=4: 1 == 0.7495601529974043 +- 0.006091813806822378
Biased estimator test for n=8: 1 == 0.867362638559428 +- 0.004629796124124494
Biased estimator test for n=16: 1 == 0.9411682174016318 +- 0.0034581022231148107
Biased estimator test for n=32: 1 == 0.9674490769406684 +- 0.002452884914118586
Biased estimator test for n=64: 1 == 0.9804292666298523 +- 0.0017597170299214334

Unbiased estimator test for n=4: 1 == 1.0102138289446236 +- 0.008225561100470576
Unbiased estimator test for n=8: 1 == 0.9942457249248356 +- 0.005346418402603227
Unbiased estimator test for n=16: 1 == 0.9967384398055572 +- 0.003638493154759803
Unbiased estimator test for n=32: 1 == 0.998204808485841 +- 0.002571835179560784
Unbiased estimator test for n=64: 1 == 0.9971881428525637 +- 0.0017733381431177173
```

Functions of averages, jackknife and bootstrap variance estimators

Consider sampling n variables $x_1, \dots, x_n \in R^m$ from a PDF $p(x)$ and the samples to be independent of each other. We define the unbiased estimator

$$\hat{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{57}$$

of

$$\bar{x} = \langle x \rangle \,. \tag{58}$$

Then a function $f : R^m \to R$ of the mean

$$f(\bar{x}) \tag{59}$$

can be estimated by

$$f(\hat{x}) \,. \tag{60}$$

If the function is linear, the estimator is also unbiased.

Next, we can relate the variance of the estimator

$$\mathrm{Var}(f, x) = \langle f(\hat{x})^2 \rangle - \langle f(\hat{x}) \rangle^2 \tag{61}$$

to the variance of $\hat{x}$ through

$$\mathrm{Var}(f, x) = \langle f(\hat{x})^2 \rangle - \langle f(\hat{x}) \rangle^2 \tag{62}$$

$$= \langle f(\bar{x} + \hat{x} - \bar{x})^2 \rangle - \langle f(\bar{x} + \hat{x} - \bar{x}) \rangle^2 \tag{63}$$

$$= \langle (f(\bar{x}) + \frac{\partial f}{\partial x_i}(\bar{x})(\hat{x} - \bar{x})_i + \frac{1}{2}\frac{\partial^2 f}{\partial x_i \partial x_j}(\bar{x})(\hat{x} - \bar{x})_i(\hat{x} - \bar{x})_j)^2 \rangle$$
$$- \langle f(\bar{x}) + \frac{\partial f}{\partial x_i}(\bar{x})(\hat{x} - \bar{x})_i + \frac{1}{2}\frac{\partial^2 f}{\partial x_i \partial x_j}(\bar{x})(\hat{x} - \bar{x})_i(\hat{x} - \bar{x})_j \rangle^2 + \langle O(\hat{x} - \bar{x})^3 \rangle \tag{64}$$

$$= (f(\bar{x})\frac{\partial^2 f}{\partial x_i \partial x_j}(\bar{x}) + \frac{\partial f}{\partial x_i}\frac{\partial f}{\partial x_j}(\bar{x}))\langle (\hat{x} - \bar{x})_i(\hat{x} - \bar{x})_j \rangle$$
$$- f(\bar{x})\frac{\partial^2 f}{\partial x_i \partial x_j}(\bar{x})\langle (\hat{x} - \bar{x})_i(\hat{x} - \bar{x})_j \rangle + \langle O(\hat{x} - \bar{x})^3 \rangle \tag{65}$$

$$= \frac{\partial f}{\partial x_i}(\bar{x})\frac{\partial f}{\partial x_j}(\bar{x})\langle (\hat{x} - \bar{x})_i(\hat{x} - \bar{x})_j \rangle + \langle O(\hat{x} - \bar{x})^3 \rangle \tag{66}$$

We have established above that due to the averaging, $\hat{x} - \bar{x} \propto 1/\sqrt{n}$.

We now first define the **single-elimination Jackknife** resamples

$$\hat{x}_j = \frac{1}{n-1}\sum_{i=1,i\neq j}^{n} x_i = \frac{1}{n-1}(n\hat{x} - x_j) = \frac{1}{n-1}((n-1)\hat{x} + \hat{x} - x_j) = \hat{x} + \frac{1}{n-1}(\hat{x} - x_j) \tag{67}$$

with which we can define the **single-elimination Jacknife variance estimator**

$$\mathrm{Var}^J(f,x) = \frac{n-1}{n}\sum_{j=1}^{n}(f(\hat{x}_j) - f(\hat{x}))^2 \tag{68}$$
$$= \frac{n-1}{n}\sum_{j=1}^{n}\left(\frac{\partial f}{\partial x_i}(\hat{x})(\hat{x}_j - \hat{x})_i\right)^2(1 + O(1/\sqrt{n})) \tag{69}$$
$$= \frac{\partial f}{\partial x_i}(\hat{x})\frac{\partial f}{\partial x_l}(\hat{x})\frac{1}{n(n-1)}\sum_{j=1}^{n}(x_j - \hat{x})_i(x_j - \hat{x})_l(1 + O(1/\sqrt{n}))\,. \tag{70}$$

We can then show that

$$\langle \mathrm{Var}^J(f,x)\rangle = \frac{\partial f}{\partial x_i}(\bar{x})\frac{\partial f}{\partial x_l}(\bar{x})\frac{1}{n}\langle\frac{1}{n-1}\sum_{j=1}^{n}(x_j - \hat{x})_i(x_j - \hat{x})_l\rangle(1 + O(1/\sqrt{n})) \tag{71}$$
$$= \frac{\partial f}{\partial x_i}(\bar{x})\frac{\partial f}{\partial x_l}(\bar{x})\frac{1}{n}\langle(x - \bar{x})(x - \bar{x})\rangle(1 + O(1/\sqrt{n})) \tag{72}$$
$$= \frac{\partial f}{\partial x_i}(\bar{x})\frac{\partial f}{\partial x_l}(\bar{x})\langle(\hat{x} - \bar{x})(\hat{x} - \bar{x})\rangle(1 + O(1/\sqrt{n})) \tag{73}$$
$$= \mathrm{Var}(f,x)(1 + O(1/\sqrt{n})) \tag{74}$$

since

$$\frac{\partial f}{\partial x_i}(\hat{x}) = \frac{\partial f}{\partial x_i}(\bar{x})(1 + O(1/\sqrt{n}))\,. \tag{75}$$

Homework: Show that this estimator is unbiased for linear f.

An alternative very popular resampling technique is the **bootstrap** method. In this method, we create $S \in N^+$ pseudosamples by randomly selecting n elements from the set of $\{x_1, \ldots, x_n\}$ with replacement. The variance over these pseudosamples then estimates the distribution variance. We illustrate both the jackknife and bootstrap methods below numerically.

In [4]:
```
def jackknife_variance_estimator(f, x):
    N = len(x)
    X = sum(x) / N
    mean = f(X)
    return sum([(f((N*X - x[j])/(N-1)) - mean)**2 for j in range(N)])*(N-1)/N

res = [np.random.normal(2.0,0.5,2) for i in range(100)]

sigma_expected = 0.5 / 2.0 * 2.0**0.5 / 100**0.5

print(jackknife_variance_estimator(lambda x: x[0]/x[1], res)**0.5 / sigma_expected)


tests_jk = [jackknife_variance_estimator(lambda x: x[0]/x[1],
                                         [np.random.normal(2.0,0.5,2) for i in range(100)])**0.5 / sigma_expected
            for l in range(100)]

print(np.mean(tests_jk), np.std(tests_jk))
```

```
0.9997201133255009
0.9943462780649956 0.068874945189182 37
```

In [5]:
```
def bootstrap_variance_estimator(f, x, S):
    N = len(x)
    X = sum(x) / N
    mean = f(X)
    return sum([ (f(sum([ random.choice(x) for l in range(N) ])/N) - mean)**2 for j in range(S) ])/S

print(bootstrap_variance_estimator(lambda x: x[0]/x[1], res, 100)**0.5 / sigma_expected)

tests_bs = [bootstrap_variance_estimator(lambda x: x[0]/x[1],
                                         [np.random.normal(2.0,0.5,2) for i in range(100)], 100)**0.5 / sigma_expected
            for l in range(100)]

print(np.mean(tests_bs), np.std(tests_bs))
```

```
1.103734908360207
0.9880356393606512 0.11291216607476749
```

Homework: repeat the calculation of the $E_1 - E_0$ energy splitting of the Harmonic oscillator using the jackknife estimator to assess the statistical uncertainty of the extracted value.

Autocorrelated data and binning

So far, we have assumed all data to be statistically independent. We have seen, however, in the last chapter that Markov chains typically yield autocorrelated data. A simple procedure to address this issue is the following.

Consider a markov chain with elements $x_1, \ldots, x_n \in R^m$. Assuming a number b that divides n, we could then average b sequential elements to a block

$$\tilde{x}_i = \frac{1}{b}\sum_{j=1}^{b} x_{ib+j}\,. \tag{76}$$

If we then define $\tilde{n} = n/b$ and

$$\hat{\hat{x}} = \frac{1}{\tilde{n}} \sum_{j=1}^{\tilde{n}} \tilde{x}_j \tag{77}$$

then

$$\langle \hat{\hat{x}} \rangle = \langle \hat{x} \rangle \tag{78}$$

and

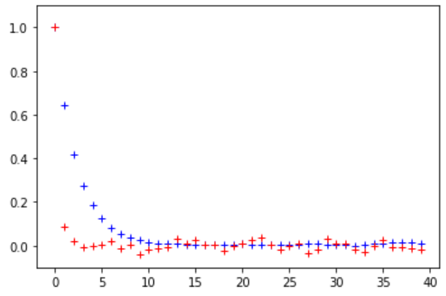
$$\langle \hat{\hat{x}}^2 \rangle = \langle \hat{x}^2 \rangle \,. \tag{79}$$

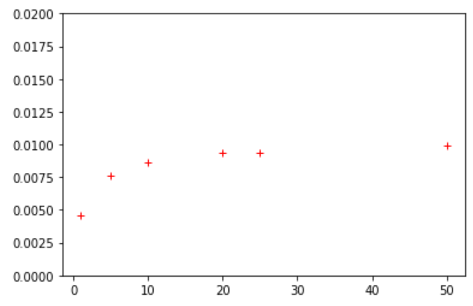
So binning does not affect the mean and variance but the binned results are generally less autocorrelated. Therefore after binning sufficiently, the data can be treated as if it were independent. If after binning too little data remains, the errors on the variance estimators, however, can grow significantly (see $1/\sqrt{n}$ terms above, n reduces by the amount of binning).

Let us demonstrate this:

In [6]:
```
def metropolis_sample_xsqr(p, step, samples):
    x0 = 0.0
    res = []
    accept = 0
    for i in range(samples):
        x = x0 + step * random.uniform(-1.0, 1.0)
        r = p(x) / p(x0)
        l = random.uniform(0.0, 1.0)
        if l < r:
            accept += 1
            x0 = x
        res.append(x0**2)
    print(f"Acceptance rate: {accept/samples}")
    return res

def bin_data(res, bin_size):
    assert len(res) % bin_size == 0
    n = len(res) // bin_size
    return [ sum(res[bin_size*i:bin_size*(i+1)])/bin_size for i in range(n) ]

def avg(X):
    return sum(X) / len(X)

res = metropolis_sample_xsqr(lambda x: np.exp(-x**2./2), 2.0, 100000)[100:]
print(avg(res))

res_binned = bin_data(res, 25)

print(avg(res_binned))

def autocovariance(Y, max_dt):
    mean = sum(Y)/len(Y)
    return [sum([ Y[t]*Y[t+dt] for t in range(len(Y) - dt) ])/(len(Y) - dt) - mean**2. for dt in range(max_dt) ]

def autocorrelator(Y, max_dt):
    Cov = autocovariance(Y, max_dt)
    return [ Cov[t] / Cov[0] for t in range(len(Cov))]


fig, ax = plt.subplots()

plt.ylim([-0.1,1.1])
ax.plot(range(40), autocorrelator(res, 40), marker='+', ls='', c='blue', label="autocorrelation original data")
ax.plot(range(40), autocorrelator(res_binned, 40), marker='+', ls='', c='red', label="autocorrelation bin-25 data")
plt.show()

bin_size = [1, 5, 10, 20, 25, 50]
error_versus_bin_size = [jackknife_variance_estimator(lambda x: x, bin_data(res, bs))**0.5 for bs in bin_size]
fig, ax = plt.subplots()
plt.ylim(0,0.02)
ax.plot(bin_size, error_versus_bin_size, marker='+', ls='', c='red', label="autocorrelation bin-25 data")
plt.show()

print(sum(res) / len(res),"+-",error_versus_bin_size[-2])
```

```
Acceptance rate: 0.62902
1.0007439855609455
1.000743985560925
```





```
1.0007439855609455 +- 0.009350224119331632
```

Combining statistically independent data

Let $x_1, \ldots, x_n$ be independent random variables, i.e., $p(x_1, \ldots, x_n) = p(x_1) \cdots p(x_n)$, then it can be shown in a similar way to the derivations above that the total variance of a function $f(x_1, \ldots, x_n)$ is the sum of the individual variances with respect to x_i assuming that the fluctuations are sufficiently small that a linearization of f is appropriate.

Bayes' theorem and finding a model that best describes a given data set

We can write the joint probability of selecting a model M and sampling a data set D as

$$P(D \cap M) = P(D|M)P(M) = P(M|D)P(D) \tag{80}$$

from which Bayes' theorem

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)} \tag{81}$$

follows. If we have no prior knowledge about which model is best, one typically uses a $P(M)$, which is independent of M. If prior knowledge was available, it can be included in a fitting procedure by appropriate choice of $P(M)$. Since also $P(D)$ is independent of M, without prior knowledge, the model which maximises $P(M|D)$ also maximises $P(D|M)$. Based on our earlier discussion of the central limit theorem, it is further reasonable to assume that $P(D|M)$ follows a normal distribution, which for a model with parameter vector $p \in R^{N_{\rm parameter}}$ can be written as

$$P(D|M) = (2\pi)^{-N_{\rm data}/2} \det(C)^{-1/2} e^{-\frac{1}{2}(D_i - M_i(p))C^{-1}_{ij}(D_j - M_j(p))}\,, \tag{82}$$

where $D \in R^{N_{\rm data}}$ and $M : R^{N_{\rm parameter}} \to R^{N_{\rm data}}$ are the model predictions for the data. This is well defined for positive definite C. Then

$$\langle D_i \rangle = M_i(p) \tag{83}$$

and $C \in R^{N_{\rm data} \times N_{\rm data}}$ with

$$\langle (D_i - \langle D_i \rangle)(D_j - \langle D_j \rangle) \rangle = C_{ij}\,. \tag{84}$$

If the individual data points are statistically independent, C must be a diagonal matrix. In our example below, this will be the case. Since C is symmetric, we can diagonalize it which yields new $N_{\rm data}$ statistically independent normal variables.

The parameters that maximize $P(D|M)$ are the same as the ones that minimize the variable

$$\chi^2 = (D_i - M_i(p))C^{-1}_{ij}(D_j - M_j(p))\,. \tag{85}$$

We call the parameters that minimize χ^2 the fit result p_0 and the value of χ^2 at the minimum χ^2_0.

It can be shown that χ^2_0 follows the PDF of a so-called χ^2-distribution with degrees of freedom $k = N_{\rm data} - N_{\rm parameter}$,

$$p(\chi^2_0) = \frac{1}{2^{k/2}\Gamma(k/2)}(\chi^2_0)^{k/2-1}e^{-\chi^2_0/2} \tag{86}$$

with mean k and variance $2k$.

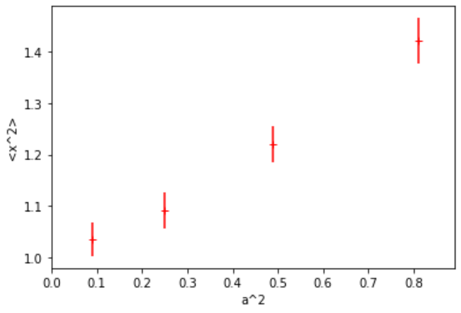
The expectation value of χ^2_0 is therefore k with fluctuations of size $\sqrt{2k}$. It is common to quantify the likelyhood of the fit by considering the P-value defined as the likelyhood that the observed χ^2_0 or a larger value would be observed, i.e.,

$$P = \int_{\chi^2_0}^{\infty} dx p(x)\,. \tag{87}$$

We now turn to a numerical demonstration based on data sampled from normal distribution with variance $f(a)$ for different parameters a. We may think of the parameter a as the lattice spacing of the path integral and $f(a)$ as the expected functional form of the continuum $(a \to 0)$ limit.

In [7]:
```python
class xsqr_data:
    def __init__(self, lattice_spacings, f_true, nbin):
        self.lattice_spacings = lattice_spacings
        self.res = [bin_data(metropolis_sample_xsqr(
            lambda x: np.exp(-x**2./2.0/f_true(a)), 2.0, 10000)[100:],nbin)
                    for a in lattice_spacings]
        self.xsqr_mean = [avg(x) for x in self.res]
        self.xsqr_err = [jackknife_variance_estimator(lambda x: x, X)**0.5 for X in self.res]

    def plot(self):
        fig, ax = plt.subplots()
        plt.xlim(0,self.lattice_spacings[-1]**2.*1.1)
        ax.errorbar([ a**2 for a in self.lattice_spacings], self.xsqr_mean, self.xsqr_err,
                    marker='+', ls='', c='red')
        plt.xlabel("a^2")
        plt.ylabel("<x^2>")
        plt.show()

random.seed(13)
d = xsqr_data([ 0.3, 0.5, 0.7, 0.9 ], lambda a: 1.0 + a**2*0.5, 25)
d.plot()
```

```
Acceptance rate: 0.6392
Acceptance rate: 0.6528
Acceptance rate: 0.6646
Acceptance rate: 0.6766
```



In [8]:

```
import scipy.optimize as opt
from scipy.integrate import quad
from scipy.special import gamma

def jackknife_covariance_estimator(f, x):
    N = len(x)
    X = sum(x) / N
    mean = f(X)
    def outer_sqr(a):
        return np.outer(a,a)
    return sum([outer_sqr(f((N*X - x[j])/(N-1)) - mean) for j in range(N)])*(N-1)/N

def get_p_value(chi2, dof):
    return quad(lambda x: 2**(-dof/2)/gamma(dof/2)*x**(dof/2-1)*np.exp(-x/2), chi2, np.inf)[0]

class model_two_parameter:
    def __init__(self, power):
        self.power = power

    def __call__(self, a, p):
        return p[0] + p[1] * a**self.power

    def parameter_gradient(self, a, p):
        return np.array([1.0,a**self.power])

    def reasonable_parameter_values(self):
        return [0.8, 0.7]

class fit:
    def __init__(self, data, lattice_spacings_to_fit, model):
        self.data = data
        self.lattice_spacings_to_fit = lattice_spacings_to_fit
        self.model = model

    def chi_square(self, data, parameter):
        return sum([ ((self.model(x, parameter) - y) / err)**2. for x, y, err in data ])

    def most_likely_parameter(self, f, data):
        opt_res = opt.minimize(lambda p: self.chi_square(data, p), self.model.reasonable_parameter_values(),
                               method = "Nelder-Mead", tol = 1e-7)
        assert opt_res.success == True
        return opt_res.x, opt_res.fun

    def select_lattices(self, data):
        return [data[i] for i in self.lattice_spacings_to_fit]

    def run(self):
        mean, chi2 = self.most_likely_parameter(
            self.model, self.select_lattices(
                list(zip(self.data.lattice_spacings, self.data.xsqr_mean, self.data.xsqr_err))
            )
        )

        covariances = []
        for i in range(len(self.data.res)):
            covariances.append(jackknife_covariance_estimator(
                lambda resi: self.most_likely_parameter(
                    self.model, self.select_lattices(
                        list(zip(self.data.lattice_spacings,
                                 self.data.xsqr_mean[:i] + [resi] + self.data.xsqr_mean[i+1:],
                                 self.data.xsqr_err))
                    )
                )[0],
                self.data.res[i]
            ))
            print("Covariance from a =",self.data.lattice_spacings[i],"is",covariances[i])

        self.best_parameter = mean
        self.best_parameter_cov = sum(covariances)
        self.chi2 = chi2
        self.dof = len(self.lattice_spacings_to_fit) - len(mean)
        self.p = get_p_value(self.chi2, self.dof)
        for i in range(len(mean)):
            print(f"parameter[{i}] = {self.best_parameter[i]} +- {self.best_parameter_cov[i][i]**0.5}")
        print(f"chi2 / dof = {self.chi2} / {self.dof}, i.e., p = {self.p}")

f = fit(d, [0,1,2,3], model_two_parameter(2))
f.run()
```

```
Covariance from a = 0.3 is [[ 0.00063662 -0.00106751]
 [-0.00106751  0.00179003]]
Covariance from a = 0.5 is [[ 0.00021218 -0.00021919]
 [-0.00021919  0.00022642]]
Covariance from a = 0.7 is [[5.32419334e-06 4.55058362e-05]
 [4.55058362e-05 3.88938018e-04]]
Covariance from a = 0.9 is [[ 0.00012678 -0.00059282]
 [-0.00059282  0.00277204]]
parameter[0] = 0.9725340439828718 +- 0.0313194600597356
parameter[1] = 0.5325144648541262 +- 0.07195438673559529
chi2 / dof = 0.645814241976392 / 2, i.e., p = 0.7240410994668618
```

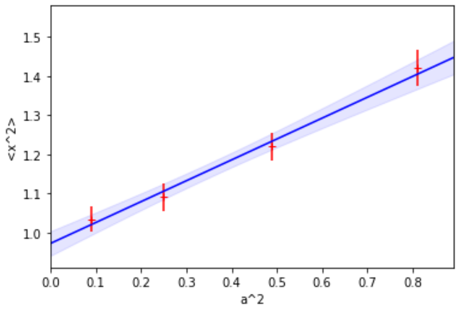
The statistical estimates of both parameters agree within errors with the function that was used to generate the data.

Plotting fit results

Once we have found the best fit parameters and have an estimate of their covariance, we can ask the question what the variance of the model prediction is for a given a value. For sufficiently small fluctuations it is straightforward to show that

$$\langle f(a,p)^2\rangle_p - \langle f(a,p)\rangle_p^2 = \frac{\partial f}{\partial p_i}(a,\langle p\rangle)\frac{\partial f}{\partial p_j}(a,\langle p\rangle)\langle (p-\langle p\rangle_p)_i (p-\langle p\rangle_p)_j\rangle\,. \tag{88}$$

In [9]:
```
def make_plot(data, fit):

    df_dp = fit.model.parameter_gradient

    def err_f(a,p,cov_p):
        return np.dot(df_dp(a,p), np.dot(cov_p, df_dp(a,p)))**0.5

    fxrange = np.arange(0.0, data.lattice_spacings[-1]*1.1, 0.01)
    fy = np.array([fit.model(asqr**0.5,fit.best_parameter) for asqr in fxrange])
    fyerr = np.array([err_f(asqr**0.5,fit.best_parameter,fit.best_parameter_cov) for asqr in fxrange])
    fig, ax = plt.subplots()
    plt.xlim(0,data.lattice_spacings[-1]**2.*1.1)
    ax.fill_between(fxrange,fy-fyerr,fy+fyerr,alpha=0.1,color="blue")
    ax.plot(fxrange,fy,c="blue")
    plt.xlabel("a^2")
    plt.ylabel("<x^2>")
    ax.errorbar([ a**2 for a in data.lattice_spacings], data.xsqr_mean, data.xsqr_err, marker='+', ls='', c='red')
    plt.show()

make_plot(d,f)
```

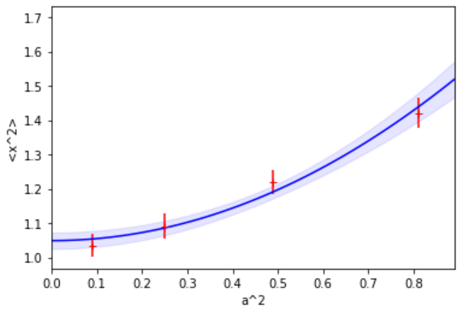


Systematic uncertainties in fits

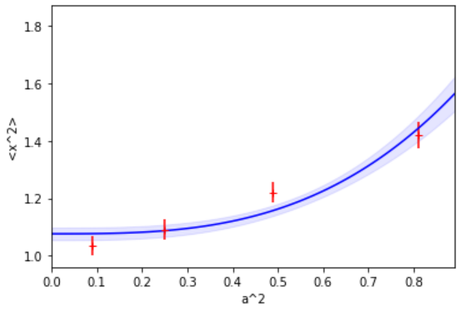
The P-value can be used to detect models that are inconsistent with the data. One typically would reject fits with a P-value below 0.05 or 0.01. Note, however, that there may be models that are not rejected based on such a criterion which would, however, yield incorrect extrapolation results. It is therefore crucial to understand from first principles, what model is expected to describe the data. Let us, e.g., vary the power parameter of the model:

In [10]:
```
f = fit(d, [0,1,2,3], model_two_parameter(4))
f.run()
make_plot(d,f)

f = fit(d, [0,1,2,3], model_two_parameter(6))
f.run()
make_plot(d,f)
```

```
Covariance from a = 0.3 is [[ 0.0002933  -0.00062207]
 [-0.00062207  0.00131937]]
Covariance from a = 0.5 is [[ 0.00018441 -0.00031133]
 [-0.00031133  0.00052559]]
Covariance from a = 0.7 is [[5.70044895e-05 6.94709773e-05]
 [6.94709773e-05 8.46638004e-05]]
Covariance from a = 0.9 is [[ 2.89871654e-05 -3.63028438e-04]
 [-3.63028438e-04  4.54648273e-03]]
parameter[0] = 1.0488704541569347 +- 0.023742388886163212
parameter[1] = 0.5927260642234137 +- 0.08047425253679717
chi2 / dof = 1.1672008310949793 / 2, i.e., p = 0.5578861244091011
```



```
Covariance from a = 0.3 is [[ 0.00021279 -0.00051691]
 [-0.00051691  0.00125567]]
Covariance from a = 0.5 is [[ 0.00016061 -0.0003554 ]
 [-0.0003554   0.00078639]]
Covariance from a = 0.7 is [[ 8.93488411e-05 -1.22643890e-05]
 [-1.22643890e-05  1.68346079e-06]]
Covariance from a = 0.9 is [[ 9.76342071e-06 -2.68073675e-04]
 [-2.68073675e-04  7.36048333e-03]]
parameter[0] = 1.0768072629936558 +- 0.02173755115720664
parameter[1] = 0.6882784657540444 +- 0.09697536912676169
chi2 / dof = 5.04266567743004 / 2, i.e., p = 0.08035243850549526
```

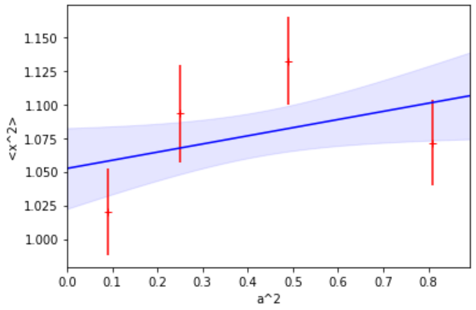


Both models still yielded an acceptable P-value, however, the a $\to 0$ result (paramter[0]) was inconsistent with the expected result of 1.

A more realistic case is that we know the asymptotic, say a $\to 0$, behavior but that at finite a there are small deviations from the fit model. Let us investigate this case. In this case, we may devise a systematic error by investigating the effect of the data points that are least expected to follow the model (here the largest values of a):

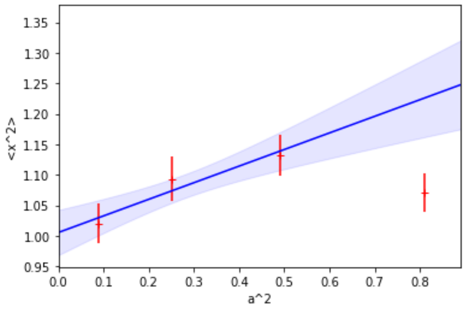
In [11]:
```
random.seed(13)

d2 = xsqr_data([ 0.3, 0.5, 0.7, 0.9 ], lambda a: 1.0 + a**2*0.5 - a**4*0.5, 25)
f2_4a = fit(d2, [0,1,2,3], model_two_parameter(2))
f2_4a.run()
make_plot(d2,f2_4a)

f2_3a = fit(d2, [0,1,2], model_two_parameter(2))
f2_3a.run()
make_plot(d2,f2_3a)

print(f"Systematic error estimate: {f2_4a.best_parameter[0] - f2_3a.best_parameter[0]}")
```

```
Acceptance rate: 0.6363
Acceptance rate: 0.6477
Acceptance rate: 0.6539
Acceptance rate: 0.635
Covariance from a = 0.3 is [[ 0.00056386 -0.00085898]
 [-0.00085898  0.00130855]]
Covariance from a = 0.5 is [[ 0.00021469 -0.00024639]
 [-0.00024639  0.00028278]]
Covariance from a = 0.7 is [[2.79959999e-05 3.76234555e-05]
 [3.76234555e-05 5.05616674e-05]]
Covariance from a = 0.9 is [[ 9.17326352e-05 -4.12311415e-04]
 [-4.12311415e-04  1.85321944e-03]]
parameter[0] = 1.0523907312375735 +- 0.029971198544555266
parameter[1] = 0.060524600513689505 +- 0.059119506270746136
chi2 / dof = 5.119393026486666 / 2, i.e., p = 0.07732820496863554
```



```
Covariance from a = 0.3 is [[ 0.00105505 -0.00245038]
 [-0.00245038  0.00569104]]
Covariance from a = 0.5 is [[ 1.69798888e-04 -1.24159148e-04]
 [-1.24159148e-04  9.07867789e-05]]
Covariance from a = 0.7 is [[ 0.00014609 -0.00103002]
 [-0.00103002  0.00726229]]
```

```
Covariance from a = 0.9 is [[0. 0.]
 [0. 0.]]
parameter[0] = 1.0053279073827692 +- 0.0370261851900245
parameter[1] = 0.27205832729781265 +- 0.11421085408373621
chi2 / dof = 0.43340017721134816 / 1, i.e., p = 0.5103252322066846
```



```
Systematic error estimate: 0.047062823854804314
```

After excluding the largest a value, the fit results agree with the expected value and the difference of a fit to three and four lattice spacings can be used as a systematic error estimate.

Understanding the exponential noise growth in correlation functions

Finally, we add a comment to the statistical noise of correlation functions. Consider the two-time correlator of the previous chapter,

$$\langle x_f x_i \rangle \, , \tag{89}$$

then its statistical variance is

$$\langle x_f^2 x_i^2 \rangle - \langle x_f x_i \rangle^2 \, . \tag{90}$$

Since both expectation values can be related to the spectrum and matrix elements of the Hamiltonian H, one can estimate the signal to noise of correlation functions in a Euclidean path integral. This is argument is typically credited to Peter Lepage.

Consider the dominant long-time energy of $\langle x_f^2 x_i^2 \rangle$ to be E_{x^2} and of $\langle x_f x_i \rangle$ to be E_x, then the noise to signal behaves as

$$\frac{\sigma^2}{\mu^2} = \frac{\langle x_f^2 x_i^2 \rangle - \langle x_f x_i \rangle^2}{\langle x_f x_i \rangle^2} \tag{91}$$

$$= e^{-(t_f - t_i)(E_{x^2} - 2E_x)} + \text{const} \, . \tag{92}$$

In the case of the Harmonic oscillator with parameters as in the last chapter, $E_{x^2} = 0$ and $E_x = 1$, so the relative noise grows exponentially!

Homework: Perform the $E_1 - E_0$ study of the Harmonic oscillator for different lattice spacings a and extrapolate to the continuum limit $a \to 0$. Provide an estimate of both statistical and systematic uncertainties.