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

Problem 1

Show that a Poisson distribution converges to a Gaussian in the large λ limit (use Stirling and first

Poisson distribution: $P(k,\lambda) = \frac{e^{-\lambda}\lambda^k}{k!}$, Gaussian distribution: $G(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$.

Write $P = P(x,\lambda)$ and take the log:

$ln(P(x)) = ln(e^{-\lambda}) + ln(\lambda^x) - ln(x!)$

Use Stirling's approximation: $n! \approx e^{-n}n^n\sqrt{2\pi n}$:

$ln(P(x)) = -\lambda + xln(\lambda) - (ln(e^{-x}x^x\sqrt{2\pi x}))$

$ln(P(x)) = -\lambda + xln(\lambda) + x - xln(x) - \frac{1}{2}ln(2\pi x)$

$ln(P(x)) = (x-\lambda) - xln(x/\lambda) - \frac{1}{2}ln(2\pi x)$

Let $y = x - \lambda$, then $\frac{x}{\lambda} = 1 + \frac{y}{\lambda}$:

$ln(P(y)) = y - (y+\lambda)ln(1+\frac{y}{\lambda}) - \frac{1}{2}ln(2\pi(y+\lambda))$

For small ϵ, the log expansion is $ln(1+\epsilon) \approx \epsilon + \frac{\epsilon^2}{2}$. Since λ is large, $\frac{y}{\lambda}$ is very small. So expand $ln(1+\frac{y}{\lambda})$:

$ln(P(y)) = y - (y+\lambda)(\frac{y}{\lambda} + \frac{y^2}{2\lambda^2}) - \frac{1}{2}ln(2\pi(y+\lambda))$

Eliminate the $1/\lambda^2$ term because it will approach zero for large λ.

$ln(P(y)) = y - (y+\lambda)(\frac{y}{\lambda}) - \frac{1}{2}ln(2\pi(y+\lambda))$

$ln(P(y)) = y - (\frac{y^2}{\lambda} + y) - \frac{1}{2}ln(2\pi(y+\lambda))$

$ln(P(y)) = \frac{y^2}{\lambda} - \frac{1}{2}ln(2\pi(y+\lambda))$

Take the exponent:

$P(y) = \frac{1}{\sqrt{2\pi\lambda}} exp(-\frac{y^2}{2\lambda})$

Sub $x = y$, $\lambda = \sigma^2$:

$P(x) = \frac{1}{\sqrt{2\pi\sigma}} exp(\frac{x^2}{2\lambda}) = G(x)$ (Gaussian centered at zero).

Problem 2

5σ is the gold standard for a believable result. Define the Gaussian approximation as "good enough" if it agrees with the Poisson distribution to within a factor of 2. How large does n need to

The premise is that $P(x = \lambda + c\lambda) = 2G(x = \lambda + c\lambda)$, where c is a constant (3 or 5).

Set the constants in the Gaussian in terms of λ: $\mu = \lambda$, $\sigma^2 = \lambda$:

Then $P(x,\lambda) = \frac{e^{\lambda}\lambda^x}{x!}$ and $2G(x,\lambda) = \frac{2}{\sqrt{2\pi\lambda}} exp(-\frac{(x-\lambda)^2}{2\lambda}) = \sqrt{\frac{2}{\pi\lambda}} exp(-\frac{(x-\lambda)^2}{2\lambda})$

Then we solve for x in $\frac{e^{\lambda}\lambda^x}{x!} = \sqrt{\frac{2}{\pi\lambda}} exp(-\frac{(x-\lambda)^2}{2\lambda})$.

Take the log of both sides and use Stirling's approximation for the $x!$.

Left hand side:

$ln(LHS) = -\lambda + xln(\lambda) - ln(x!)$

$ln(LHS) = -\lambda + xln(\lambda) - (-x + xln(x) + \frac{1}{2}ln(2\pi x))$

$ln(LHS) = x - \lambda + xln(\frac{\lambda}{x}) - \frac{1}{2}ln(2\pi x)$

Right hand side:

$ln(RHS) = \frac{1}{2}ln(\frac{2}{\pi\lambda}) - \frac{(x-\lambda)^2}{\sqrt{2\lambda}}$.

The x (or n) where $ln(LHS) = ln(RHS)$ is where these two curves meet, evaluated at

In [2]:
```python
# define functions for ln(LHS) and ln(RHS) from the above equations:

def log_lhs(x,L):
    # returns log(Poisson(x,lambda))
    return x - L + x*np.log(L/x) - 0.5*np.log(2*x*np.pi)

def log_rhs(x,L):
    # returns log(2*Gaussian(x,lambda))
    return 0.5*np.log(2/(L*np.pi))  - ((x-L)**2/(2*L))

# evaluate by taking x-->lamda+3*sqrt(lambda)
L = np.linspace(0.1,10,1001)
X3 = L + 3*np.sqrt(L)
log_lhs_3 = log_lhs(X3,L)
log_rhs_3 = log_rhs(X3,L)

# find where sign of the difference between the right and left hand
# sides change to determine n. This is where Poisson and Gaussian cross
start_sign = np.sign(log_lhs_3[0]-log_rhs_3[0])
# condition for if max lambda (L) isn't big enough:
if np.sign(log_lhs_3[-1]-log_rhs_3[-1]) == start_sign:
    print('extend L for 3 sigma')
for i in np.arange(1,len(X3)):
    if np.sign(log_lhs_3[i]-log_rhs_3[i]) != start_sign:
        n3 = X3[i]
        print(r'Poisson = Gaussian when n = {} for 3 sigma'.format(int(X
3[i])))
        break

# plot the results
plt.figure()
plt.plot(X3, log_lhs_3, label='log(lhs)')
plt.plot(X3, log_rhs_3, label='log(rhs)')
plt.axvline(n3, c='k', label='n')
plt.legend()
plt.grid()

# Repeat:

# evaluate by taking x-->lambda+5*sqrt(lambda)
L = np.linspace(0.1,600,1001)
X5 = L + 5*np.sqrt(L)
log_lhs_5 = log_lhs(X5,L)
log_rhs_5 = log_rhs(X5,L)

# find where sign of the difference between the right and left hand
# sides change to determine n. This is where Poisson and Gaussian cross
start_sign = np.sign(log_lhs_5[0]-log_rhs_5[0])
# condition for if max lambda (L) isn't big enough:
if np.sign(log_lhs_5[-1]-log_rhs_5[-1]) == start_sign:
    print('extend L for 5 sigma')
for i in np.arange(1,len(X5)):
    if np.sign(log_lhs_5[i]-log_rhs_5[i]) != start_sign:
        n5 = X5[i]
        print(r'Poisson = Gaussian when n = {} for 5 sigma'.format(int(X
5[i])))
        break

# plot results
plt.figure()
plt.plot(X5, log_lhs_5, label='log(lhs)')
plt.plot(X5, log_rhs_5, label='log(rhs)')
plt.axvline(n5, c='k', label='n')
plt.legend()
plt.grid()
```

```
Poisson = Gaussian when n = 17 for 3 sigma
Poisson = Gaussian when n = 696 for 5 sigma
```

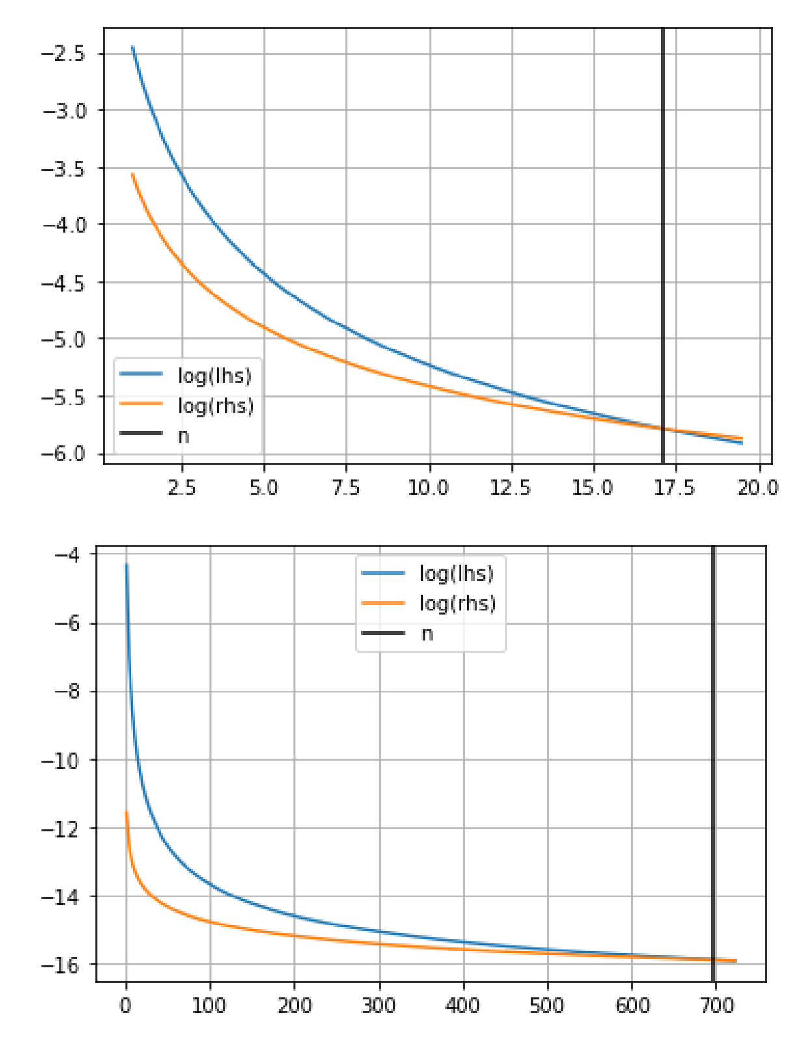

Fig: The Gaussian distribution is in agreement with the Poisson distribution within 5σ when $n = 696$. The Gaussian is in agreement with Poisson within 3σ when $n = 17$.

Problem 3

n Gaussian-distributed data points with identical σ and unkown mean. What is the error on the maximum-likelihood estimate of the mean? If half the data is wrong by a factor of $\sqrt{2}$ (variance is off by a factor of 2), what is the true error on the estimated mean, compare it to maximum-likelihood with correct noise estimates. What if 1% of the data is underweighted by a factor of

This derivation follows from Worked Example 2 in Lecture 2.

For n Gaussian-distributed data points we have the model m, data d, weights w, we have $m = \frac{\sum w_i d_i}{\sum w_i d_i}$ and the variance is $\frac{1}{\sum w_i}$. (Sums are from $i = 1$ to N)

We define the weights as $w_i = \frac{1}{\sigma_i^2}$.

The error (E) is the square root of the variance:

$$E = \sqrt{\frac{1}{\sum w_i}} = \sqrt{\frac{1}{\sum \frac{1}{\sigma_i^2}}}$$

Since $\sigma_i = \sigma$ (same standard deviation for all data points):

$\sum_i^N \frac{1}{\sigma_i^2} = N\frac{1}{\sigma^2}$.

The error is then:

$E = \sqrt{\frac{\sigma^2}{N}} = \frac{\sigma}{\sqrt{N}}$.

If half of the data are wrong, then the error is:

$$E = (\sum_i^{N/2} \frac{1}{\sigma_i^2} + \sum_i^{N/2} \frac{1}{2\sigma_i^2})^{-1/2} = (\frac{N}{2\sigma^2} + \frac{N}{2\sigma^2})^{-1/2} = (\frac{6N}{8\sigma^2})^{-1/2} = \frac{2\sigma}{\sqrt{3N}} \approx 1.15\frac{\sigma}{\sqrt{N}}$$

This is about 15% off from the error on the maximum-likelihood estimated mean.

If 1% of the data are underweighted by a factor of 100, this means $w_i \rightarrow w_i/100$ or $\sigma_i \rightarrow 100\sigma_i$. The error is then:

$$E = (\sum_i^{N/100} \frac{1}{100\sigma_i^2} + \sum_i^{99N/100} \frac{1}{\sigma_i^2})^{-1/2} = (\frac{N}{100^2\sigma^2} + \frac{99N}{100\sigma^2})^{-1/2} = (\frac{9901}{10000})^{-1/2}\frac{\sigma}{\sqrt{N}} \approx 1.00$$

This is about 0.5% off from the error on the maximum-likelihood estimated mean.

If 1% of the data are overweighted by a factor of 100, this means $w_i \rightarrow 100w_i$ or $\sigma_i \rightarrow \frac{\sigma_i}{100}$. The error is then:

$$E = (\sum_i^{N/100} \frac{100}{\sigma_i^2} + \sum_i^{99N/100} \frac{1}{\sigma_i^2})^{-1/2} = (\frac{100N}{100\sigma^2} + \frac{99N}{100\sigma^2})^{-1/2} = (\frac{199}{100})^{-1/2}\frac{\sigma}{\sqrt{N}} \approx 0.701\frac{\sigma}{\sqrt{N}}$$

This is about 30% off from the error on the maximum-likelihood estimated mean.

In [3]:
```python
# do the number math:
2/np.sqrt(3), (9901/10000)**-0.5, (199/100)**-0.5
```

Out[3]:
```
(1.1547005383792517, 1.004987059618685, 0.708881205008336)
```

Problem 4

Write a program that generates Gaussian noise + a signal (unit Gaussian). Estimate noise by assuming it's constant within a chunk and equal to the scatter in the observed data. Show that the least-squares estimate for one chunk is unbiased, but least-squares estimate for many chunks is biased LOW. Fit an amplitude and error to each chunk and use that to get an overall amplitude

In [4]:
```python
# set number of points, size of individual chunks, size of multiple chun
ks
npts, chunk_size, chunk_size_new = 51,3,17
# this was for a test with more points:
#npts, chunk_size, chunk_size_new = 10000,10,500

# define x-values
x = np.linspace(-5,5,npts)

# generate the signal+noise
data = np.exp(-x**2/2) + np.random.randn(len(x))

# define size of chunks
nchunks = int(len(x)/chunk_size)
# print a sanity check:
#print('individual chunks have size = {}, for a total of {} chunks. Tota
l points is {}={}'.format(chunk_size,nchunks,npts, chunk_size*nchunks))

# divide x into chunks
x_chunks = [x[chunk_size*i:chunk_size*(i+1)] for i in range(nchunks)]
data_chunks = [data[chunk_size*i:chunk_size*(i+1)] for i in range(nchunk
s)]

# estimate constant noise for each chunk
noise_chunk = [np.std(data_chunks[i]) for i in range(nchunks)]
# repeat so that each data point has an associate noise
noise = np.repeat(noise_chunk,chunk_size)

# least-squares estimate for each chunk (should be unbiased):
amp_fits, err_fits = np.zeros(nchunks), np.zeros(nchunks)
for i in range(nchunks):
    # take one chunk of data:
    patch = data_chunks[i]
    # model is the template signal:
    mymodel = np.exp(-patch**2/2)
    # the least-squares best-fit amplitude for constant noise:
    amp_fit = np.sum(mymodel*patch)/np.sum(mymodel**2)
    amp_fits[i] = amp_fit
    # calculate the error (scatter between data and the model):
    err_fit = np.std(patch - mymodel*amp_fit)
    err_fits[i] = err_fit

# least-squares estimate for multiple chunks (should be biased low):
nchunks_new = int(len(x)/chunk_size_new) # look at more chunks
# print a sanity check:
#print('multiple chunks have size = {}, for a total of {} chunks. Total
 points is {}={}'.format(chunk_size_new,nchunks_new,npts, chunk_size_new
*nchunks_new))
amp_fits_multip, err_fits_multip = np.zeros(nchunks_new), np.zeros(nchun
ks_new)
for i in range(nchunks_new):
    patch = data[chunk_size_new*i:chunk_size_new*(i+1)]
    # model is the template signal:
    mymodel = np.exp(-patch**2/2)
    # get the least-squares best-fit amplitude for constant noise:
    amp_fit = np.sum(mymodel*patch)/np.sum(mymodel**2)
    amp_fits_multip[i] = amp_fit
    # calculate the error (scatter between data and the model):
    err_fit = np.std(patch - mymodel*amp_fit)
    err_fits_multip[i] = err_fit

# use weighted average to get overall amplitude and error:
amp_avg = np.sum(amp_fits)/len(amp_fits)
err_avg = np.sum(err_fits)/len(err_fits)

# plot the results
plt.figure()
plt.scatter(amp_fits, err_fits, marker='.', label='individual chunks')
plt.scatter(amp_fits_multip, err_fits_multip, marker='.', label='multipl
e chunks')
plt.scatter(amp_avg, err_avg, marker='o', label='average')
plt.ylabel('fitted error')
plt.x
plt.l
```

Out[4]:
```
<matplotlib.legend.Legend at 0x7fcc6adee850>
```

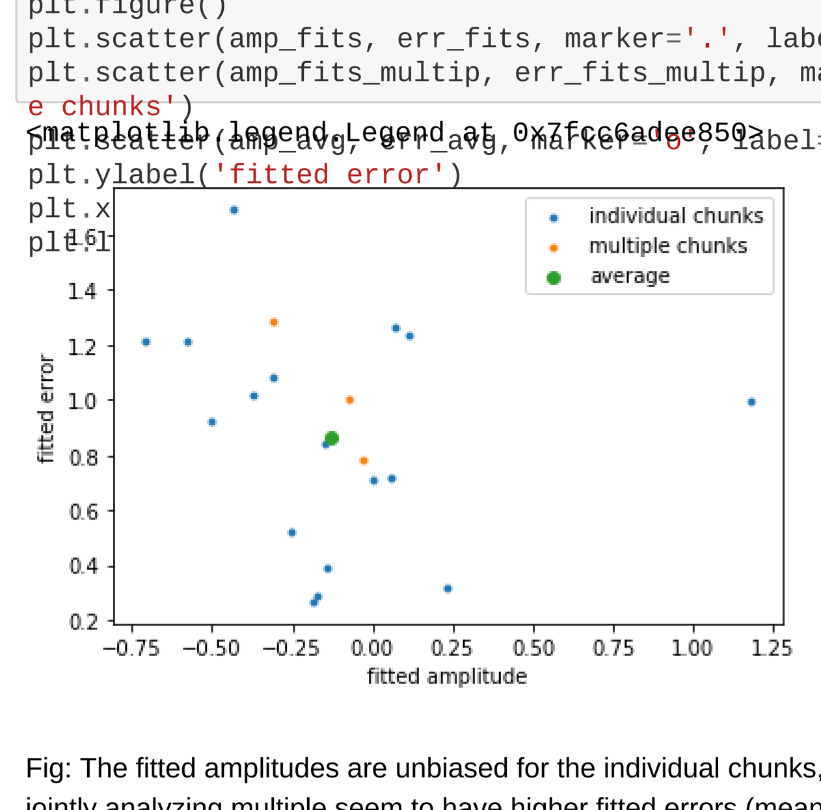

Fig: The fitted amplitudes are unbiased for the individual chunks, while the fitted amplitudes from jointly analyzing multiple seem to have higher fitted errors (meaning this analysis is biased high? I

Problem 5

Show that in rotation space, $N_{ij} =< n_i n_j >$.

Original expression for χ^2 is $\chi^2 = n^T N^{-1} n$, where n are the residuals between the model and the data: $n = a - A(m)$, and N is diagonal matrix with $N_{ii} =< n_i^2 >$.

Introduce orthogonal rotation matrix V where $VV^T = V^TV = I$, which does not change χ^2:

$\chi^2 = n^tV^TVN^{-1}V^TVn = (Vn)^T(VNV^T)^{-1}(Vn)$.

Let $Vr = \tilde{n}$ (so $(Vr)^T = \tilde{n}$), $VNV^T = \tilde{N}$:

$\chi^2 = \tilde{n}\tilde{N}^{-1}\tilde{}n$, where now $\tilde{N}$ is not diagonal.

Use $\tilde{n}_i = V_{:,i}^T n$ and $\tilde{n}_j = n^T V_{:,j}$ to find $< \tilde{n}_i\tilde{n}_j >$:

$< \tilde{n}_i\tilde{n}_j >=< V_{:,i}^T nn^T V_{:,j} >$.

Since $nn^T = n^2$ and $< nnT >= N$: $< V_{:,i}^T nn^T V_{:,j} >= V_{:,i}^T N V_{:,j} = \tilde{N}_{ij}$.

Therefore $< \tilde{n}_i\tilde{n}_j >= \tilde{N}_{ij}$.

In [ ]: