Problem Set 1 (PHYS 641) Matias Castro Tapia

Problem 1

The Poissan distribution $P(K|\lambda) = \frac{\lambda^K e^{-\lambda}}{K!}$. Using the Stirling approximation considerig $K \gg 1$:

$K! \sim \sqrt{2\pi K}\left(\frac{K}{e}\right)^K \Rightarrow P(K|\lambda) \simeq \frac{\lambda^K e^{-\lambda}}{\sqrt{2\pi K}\left(\frac{K}{e}\right)^K}$ And now we can take the log.

$$\Rightarrow \ln(P) = K\ln(\lambda) - \lambda - K\ln(K) + K - \ln(\sqrt{2\pi K}) = -K\ln\left(\frac{K}{\lambda}\right) - \lambda + K - \ln(\sqrt{2\pi K})$$

Considering the following change of variable $y = K - \lambda$

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

If $y \ll \lambda$ and we consider the Taylor Series of $\ln(1+x)$ around 0:

$$\ln\left(1+\frac{y}{\lambda}\right) \simeq \frac{y}{\lambda} - \frac{y^2}{2\lambda^2} + \frac{y^3}{3\lambda^3} - \dots \Rightarrow \ln(P) \simeq -(\lambda+y)\left(\frac{y}{\lambda} - \frac{y^2}{2\lambda^2}\right) + y - \ln\left(\sqrt{2\pi(\lambda+y)}\right)$$

$\hookrightarrow$ taking $\frac{y^3}{\lambda^3} \sim 0$

$$\Rightarrow -y + \frac{y^2}{2\lambda} - \frac{y^2}{\lambda} + \frac{y^3}{2\lambda^2} + y - \ln\left(\sqrt{2\pi(\lambda+y)}\right) \simeq \ln(P)$$

Considering $y \ll \lambda$ again $\frac{y^3}{2\lambda^2} \sim 0$ and $2\pi(\lambda+y) \simeq 2\pi\lambda$

$$\Rightarrow \ln(P) \simeq \frac{-y^2}{2\lambda} - \ln(\sqrt{2\pi\lambda}) = -\frac{(K-\lambda)^2}{2\lambda} - \ln(\sqrt{2\pi\lambda})$$

$$\Rightarrow P(K|\lambda) \simeq \frac{e^{-\frac{(K-\lambda)^2}{2\lambda}}}{\sqrt{2\pi\lambda}}$$

We know that the standard deviation of a Poisson distribution is $\sigma = \sqrt{\lambda}$

$$\Rightarrow P(K|\lambda,\sigma) \simeq \frac{e^{-\frac{(K-\lambda)^2}{2\sigma^2}}}{\sqrt{2\pi}\,\sigma}$$

Then, the Poisson distribution converges to a Gaussian distribution with mean λ and standard deviation $\sigma = \sqrt{\lambda}$ for a large value of λ.

Problem 2

Considering the Gaussian distribution $G(x|\mu,\sigma) = \dfrac{e^{\frac{-(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\,\sigma}$, we know that the maximum of the distribution is reached when $x=\mu$:

$G_{max}(\mu|\mu,\sigma) = \dfrac{e^0}{\sqrt{2\pi}\,\sigma} = \dfrac{1}{\sqrt{2\pi}\,\sigma}$ and $\sigma=\sqrt{\mu}$ if we consider the Gaussian as an approximation of a Poisson distribution

$\Rightarrow G_{max} = \dfrac{1}{\sqrt{2\pi\mu}}$

On the other hand:

$G_{5\sigma}(\mu+5\sigma|\mu,\sigma) = \dfrac{e^{\frac{-(\mu+5\sigma-\mu)^2}{2\sigma^2}}}{\sqrt{2\pi}\,\sigma} = \dfrac{e^{-25/2}}{\sqrt{2\pi}\,\sigma} \Rightarrow \dfrac{G_{5\sigma}}{G_{max}} = e^{-\frac{25}{2}} \simeq 4\times10^{-6}$

$\hookrightarrow \sigma=\sqrt{\mu}$ for Poisson approx.

Then, at $\mu+5\sigma$ the distribution reaches $4\times10^{-4}\,\%$ of the maximum.

Now, we can consider the Poisson distribution $P(x|\mu) = \dfrac{\mu^x e^{-\mu}}{x!}$.

The maximum of the Poisson distribution is reached at $x=\mu$

$\Rightarrow P_{max}(\mu|\mu) = \dfrac{\mu^\mu e^{-\mu}}{\mu!}$; and for $\mu+5\sigma \Rightarrow P_{5\sigma}(\mu+5\sigma|\mu) = \dfrac{\mu^{5\sigma+\mu}e^{-\mu}}{(5\sigma+\mu)!}$

$\Rightarrow \dfrac{P_{5\sigma}}{P_{max}} = \dfrac{\mu^{5\sigma}\mu!}{(5\sigma+\mu)!} = \dfrac{\mu^{5\sqrt{\mu}}\mu!}{(5\sqrt{\mu}+\mu)!} \simeq \dfrac{\mu^{5\sqrt{\mu}}\sqrt{2\pi\mu}\left(\frac{\mu}{e}\right)^\mu}{\sqrt{2\pi(\mu+5\sqrt{\mu})}\left(\frac{\mu+5\sqrt{\mu}}{e}\right)^{\mu+5\sqrt{\mu}}}$

for $\sigma=\sqrt{\mu}$

for a μ large enough and considering the Stirling approximation.

$\Rightarrow \dfrac{P_{5\sigma}}{P_{max}} \simeq \left(\dfrac{e\mu}{\mu+5\sqrt{\mu}}\right)^{5\sqrt{\mu}}\left(\dfrac{\mu}{\mu+5\sqrt{\mu}}\right)^{\mu+\frac{1}{2}}$ Then, $\dfrac{P_{5\sigma}}{P_{max}}$ must be at most $2e^{-\frac{25}{2}}$ to consider that the Gaussian and the Poisson distributions agree within a factor of 2 at 5σ.

$\hookrightarrow$ to find a μ large enough I solved $\dfrac{P_{5\sigma}}{P_{max}} = 2e^{-25/2}$ using a plot in python

For 3σ we can find $\dfrac{G_{3\sigma}}{G_{max}} = e^{-9/2}$

and $\dfrac{P_{3\sigma}}{P_{max}} \approx \left(\dfrac{e\mu}{\mu+3\sqrt{\mu}}\right)^{3\sqrt{\mu}}\left(\dfrac{\mu}{\mu+3\sqrt{\mu}}\right)^{\mu+1/2}$

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
```

I defined prop5s as the routine for the proportion $P_{5\sigma}/P_{max}$ and prop3s for the proportion $P_{3\sigma}/P_{max}$

In [2]:

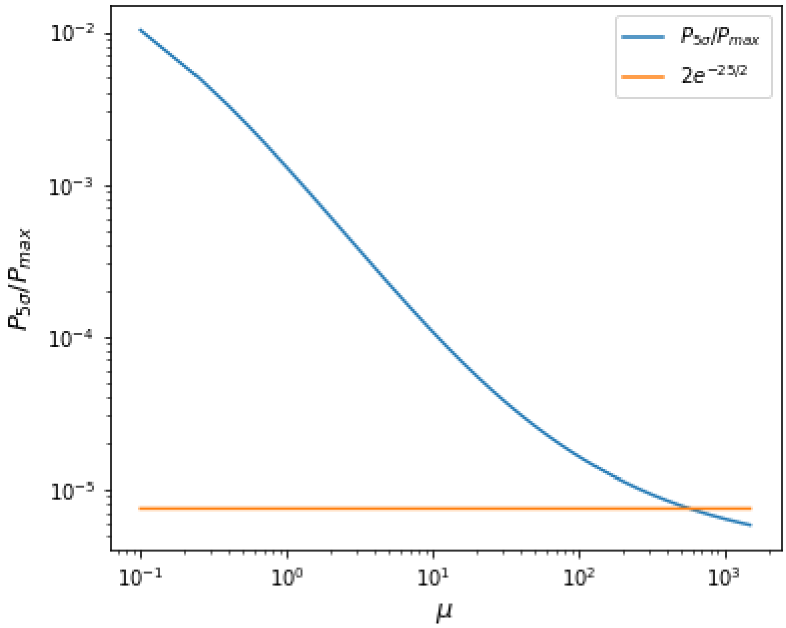
```
def prop5s(mu):
    return ((np.e*mu/(mu+5*np.sqrt(mu)))**(5*np.sqrt(mu)))*((mu/(mu+5*np.sqrt(mu)))**(mu+0.
```

In [3]:

```
x=np.linspace(0.1,1500,10001)
```

In [4]:

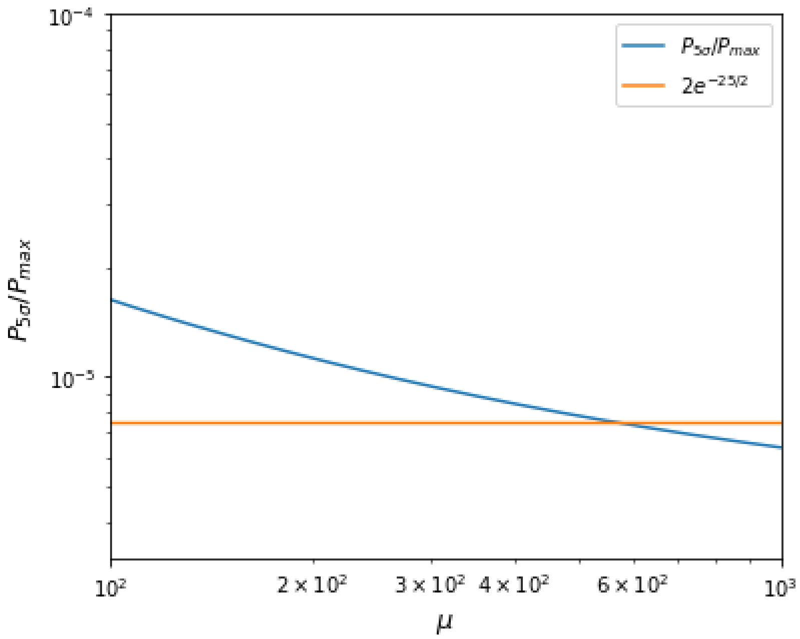
```
plt.figure(figsize=(6,5))
plt.loglog(x,prop5s(x),label=r'$P_{5\sigma}/P_{max}$')
plt.loglog(x,2*np.exp(-25/2)*np.ones(len(x)),label=r'$2e^{-25/2}$')
plt.xlabel(r'$\mu$',fontsize=13)
plt.ylabel(r'$P_{5\sigma}/P_{max}$',fontsize=13)
plt.legend()
plt.show()
```



I can note that for $\mu \approx 1000$ the proportion is about $2e^{-25/2}$ then for $x = \mu + 5\sqrt{\mu} \approx 1000 + 5\sqrt{1000} \approx 1200$ data points the Gaussian distribution will be a good enough approximation for the Poisson distribution as they agree within a factor of 2 at 5σ. If we look closer we can estimate $\mu \approx 600$ and $x = \mu + 5\sqrt{\mu} \approx 600 + 5\sqrt{600} \approx 723$

In [5]:

```
plt.figure(figsize=(6,5))
plt.loglog(x,prop5s(x),label=r'$P_{5\sigma}/P_{max}$')
plt.loglog(x,2*np.exp(-25/2)*np.ones(len(x)),label=r'$2e^{-25/2}$')
plt.xlabel(r'$\mu$',fontsize=13)
plt.ylabel(r'$P_{5\sigma}/P_{max}$',fontsize=13)
plt.xlim(10**2,10**3)
plt.ylim(10**-5.5,10**-4)
plt.legend()
plt.show()
```



I also defined an analytical form of the Gaussian and the Poisson distribution to check how they look for our estimation.

In [11]:

```
n=np.array([i for i in range(1501)])
```

In [12]:

```
def log_factorial(x):
    return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
```

In [13]:

```
def Poisson(x,mu):
    return np.exp(x*np.log(mu)-mu-log_factorial(x))
```

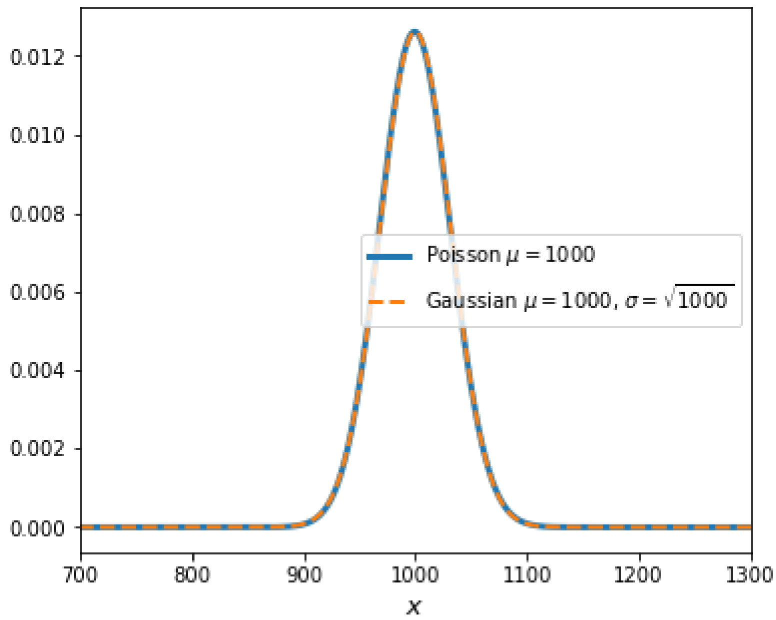
In [14]:

```
def Gaussian(x,mu,s):
    return np.exp((-(x-mu)**2)/(2*s**2))/(np.sqrt(2*np.pi)*s)
```

In [27]:

```
plt.figure(figsize=(6,5))
plt.plot(n,Poisson(n,1000),label=r'Poisson $\mu=1000$',linewidth=3)
plt.plot(n,Gaussian(n,1000,np.sqrt(1000)),'--',
         label=r'Gaussian $\mu=1000$, $\sigma=\sqrt{1000}$',linewidth='2')
plt.xlabel(r'$x$',fontsize=13)
plt.xlim(700,1300)
plt.legend()
plt.show()
```

```
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: divide by zero encountered in log
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: invalid value encountered in multiply
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
```



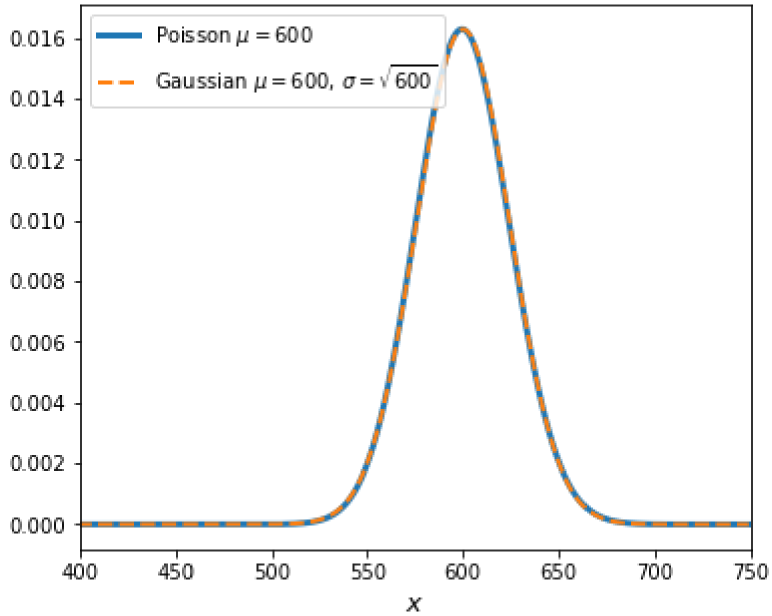
In [17]:

```
n2=np.array([i for i in range(750)])
```

In [28]:

```
plt.figure(figsize=(6,5))
plt.plot(n2,Poisson(n2,600),label=r'Poisson $\mu=600$',linewidth=3)
plt.plot(n2,Gaussian(n2,600,np.sqrt(600)),'--',
         label=r'Gaussian $\mu=600$, $\sigma=\sqrt{600}$',linewidth='2')
plt.xlabel(r'$x$',fontsize=13)
plt.xlim(400,750)
plt.legend()
plt.show()
```

```
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: divide by zero encountered in log
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: invalid value encountered in multiply
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
```



Below I did the same for 3σ

In [23]:

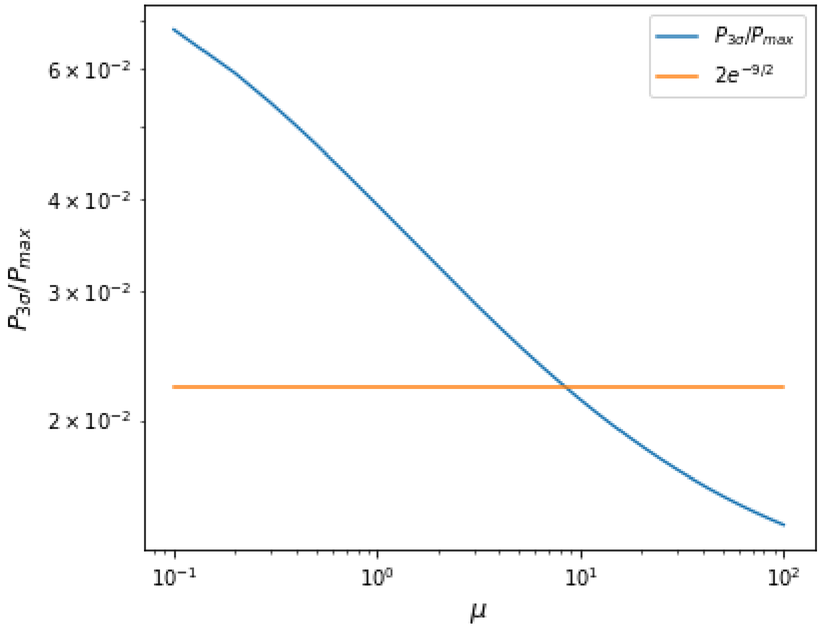
```
def prop3s(mu):
    return ((np.e*mu/(mu+3*np.sqrt(mu)))**(3*np.sqrt(mu))*((mu/(mu+3*np.sqrt(mu)))**(mu+0.
```

In [24]:

```
xx=np.linspace(0.1,100,1001)
```

In [29]:

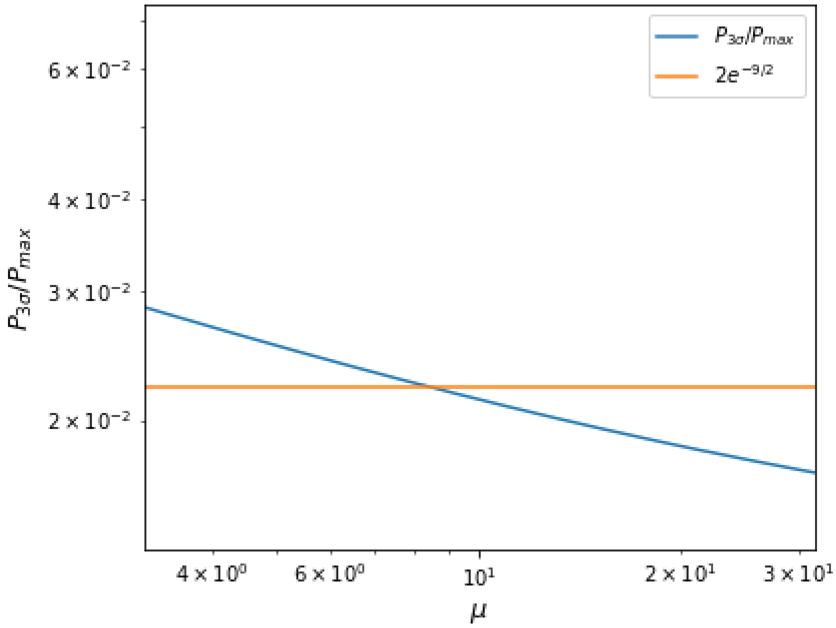
```
plt.figure(figsize=(6,5))
plt.loglog(xx,prop3s(xx),label=r'$P_{3\sigma}/P_{max}$')
plt.loglog(xx,2*np.exp(-9/2)*np.ones(len(xx)),label=r'$2e^{-9/2}$')
plt.xlabel(r'$\mu$',fontsize=13)
plt.ylabel(r'$P_{3\sigma}/P_{max}$',fontsize=13)
plt.legend()
plt.show()
```



I can note that for $\mu \approx 10$ the proportion is about $2e^{-9/2}$ then for $x = \mu + 3\sqrt{\mu} \approx 10 + 3\sqrt{10} \approx 20$ data points the Gaussian distribution will be a good enough approximation for the Poisson distribution as they agree within a factor of 2 at 3σ. If we look closer we can estimate $\mu \approx 9$ and $x = \mu + 3\sqrt{\mu} \approx 9 + 3\sqrt{9} = 18$

In [31]:

```
plt.figure(figsize=(6,5))
plt.loglog(xx,prop3s(xx),label=r'$P_{3\sigma}/P_{max}$')
plt.loglog(xx,2*np.exp(-9/2)*np.ones(len(xx)),label=r'$2e^{-9/2}$')
plt.xlabel(r'$\mu$',fontsize=13)
plt.ylabel(r'$P_{3\sigma}/P_{max}$',fontsize=13)
plt.xlim(10**0.5,10**1.5)
#plt.ylim(10**-2,10**-2.5)
plt.legend()
plt.show()
```



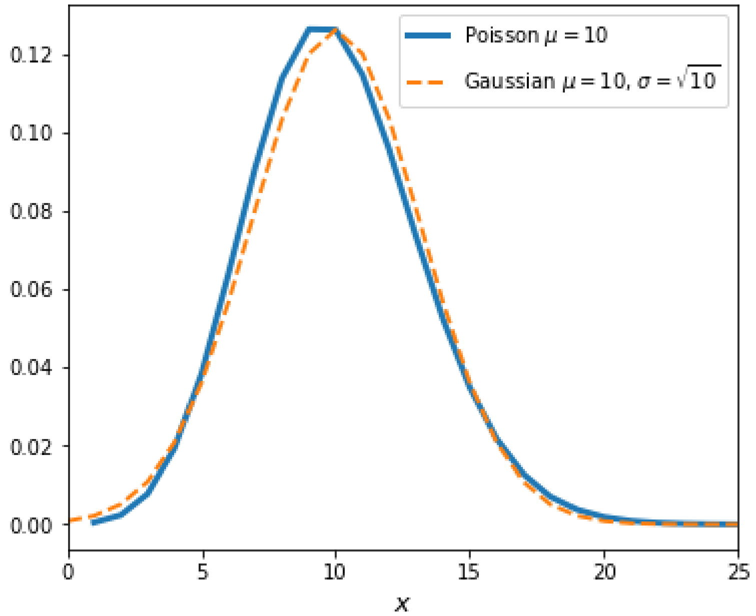
In [32]:

```
nn=np.array([i for i in range(31)])
```

In [37]:

```
plt.figure(figsize=(6,5))
plt.plot(nn,Poisson(nn,10),label=r'Poisson $\mu=10$',linewidth=3)
plt.plot(nn,Gaussian(nn,10,np.sqrt(10)),'--',
         label=r'Gaussian $\mu=10$, $\sigma=\sqrt{10}$',linewidth='2')
plt.xlabel(r'$x$',fontsize=13)
plt.xlim(0,25)
plt.legend()
plt.show()
```

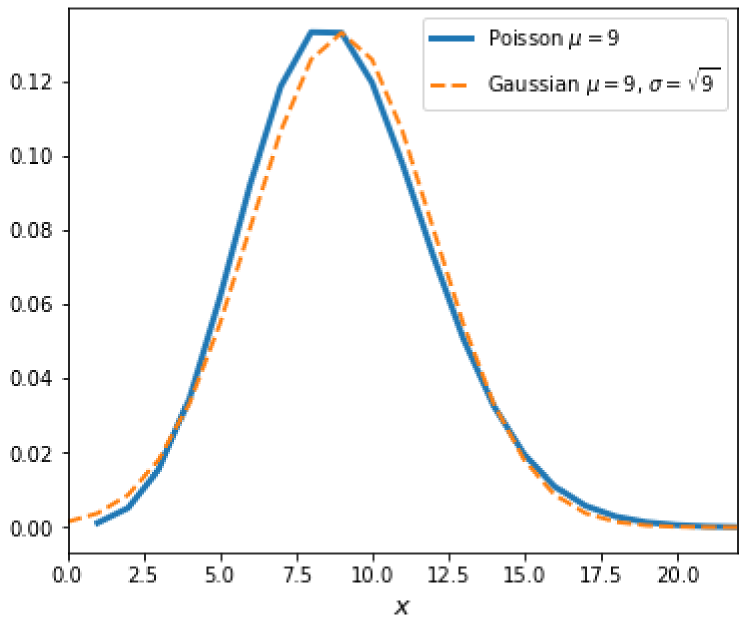
```
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: divide by zero encountered in log
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: invalid value encountered in multiply
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
```



In [38]:

```
plt.figure(figsize=(6,5))
plt.plot(nn,Poisson(nn,9),label=r'Poisson $\mu=9$',linewidth=3)
plt.plot(nn,Gaussian(nn,9,np.sqrt(9)),'--',
         label=r'Gaussian $\mu=9$, $\sigma=\sqrt{9}$',linewidth='2')
plt.xlabel(r'$x$',fontsize=13)
plt.xlim(0,22)
plt.legend()
plt.show()
```

```
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: divide by zero encountered in log
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
C:\Users\Odette\AppData\Local\Temp\ipykernel_7304\1206770816.py:2: RuntimeWa
rning: invalid value encountered in multiply
  return np.log(np.sqrt(2*np.pi*x))+x*(np.log(x/np.e))
```



Problem 3

Let's define the following matrix $N = \begin{pmatrix} \sigma_1^2 & 0 & \cdots \\ 0 & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_n^2 \end{pmatrix}$ for uncorrelated noises; then, for the n Gaussian-distributed points we have:

$$\chi^2 = (\vec{x}-\vec{\mu})^T N^{-1} (\vec{x}-\vec{\mu}) = (x_1-\mu_1 \;\; x_2-\mu_2 \;\ldots\; x_n-\mu_n) \begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & \frac{1}{\sigma_n^2} \end{pmatrix} \begin{pmatrix} x_1-\mu_1 \\ x_2-\mu_2 \\ \vdots \\ x_n-\mu_n \end{pmatrix}$$

Then, if $\sigma_i = \sigma$ and $\mu_i = \mu$:

$$\chi^2 = \sum_{i=1}^{n} \frac{(x_i-\mu)^2}{\sigma^2}$$ $\Rightarrow$ To find the maximum likelihood of μ we have to minimize χ^2 $\Rightarrow \frac{\partial \chi^2}{\partial \mu} = -2\sum_{i=1}^{n} \frac{(x_i-\mu)}{\sigma^2} = 0$

$$\Rightarrow \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \mu = 0 \Rightarrow \sum_{i=1}^{n} x_i = n\mu \Rightarrow \mu = \frac{\sum_{i=1}^{n} x_i}{n}$$

As $\chi^2 = (\vec{x}-A\vec{m})^T N^{-1} (\vec{x}-A\vec{m}) \Rightarrow \vec{m} = \vec{\mu} \Rightarrow A = \mathbb{I}$ identity matrix

we obtained that the error in the maximum likelihood can be obtained from:

$$\langle \vec{m}\vec{m}^T \rangle = \langle (A^T N^{-1} A)^{-1} \rangle = \sigma_\mu^2$$

$$\Rightarrow \sigma_\mu^2 = \langle (N^{-1})^{-1} \rangle = \frac{\sum_{i=1}^{n} \sigma_i^2 \left(\frac{1}{\sigma_i^2}\right)}{\sum_{i=1}^{n} \left(\frac{1}{\sigma_i^2}\right)} = \frac{1}{\sum_{i=1}^{n} \left(\frac{1}{\sigma_i^2}\right)} \overset{\sigma_i=\sigma}{=} \frac{1}{\left(\frac{n}{\sigma^2}\right)} = \frac{\sigma^2}{n}$$

$$\Rightarrow \sigma_\mu = \frac{\sigma}{\sqrt{n}}$$ } error in the max. likelihood estimation

If we obtain the errors wrong on half the data by a factor of $\sqrt{2}$:

$$N_* = \begin{pmatrix} \sigma_1^2 & & \\ & \sigma_2^2 & \\ & & \ddots \\ & & & \sigma_n^2 \end{pmatrix} \Rightarrow N_{*ij} = \begin{cases} \delta_{ij}\sigma^2 & \text{for } i,j \le n/2 \\ \delta_{ij} 2\sigma^2 & \text{for } i,j > n/2 \end{cases}$$

$$\Rightarrow \sigma_{\mu_1}^2 = \frac{1}{\sum_{i=1}^{n}\left(\frac{1}{\sigma_i^2}\right)} = \frac{1}{\sum_{i=1}^{n/2}\left(\frac{1}{\sigma^2}\right) + \sum_{i=\frac{n}{2}+1}^{n}\left(\frac{1}{2\sigma^2}\right)} = \frac{1}{\frac{n}{2\sigma^2} + \frac{n}{4\sigma^2}} = \frac{4}{3}\frac{\sigma^2}{n}$$

$$\Rightarrow \sigma_{\mu_1} = \frac{2}{\sqrt{3}}\frac{\sigma}{\sqrt{n}} = \frac{2}{\sqrt{3}}\sigma_\mu > \sigma_\mu$$ } This error is larger than that we obtained for the case where all σs were identical.

Now, if we underweight 1% of the data by a factor of ~100:

$$\sigma_{\mu_2}^2 = \left(\sum_{i=1}^{n}(1/\sigma_i^2)\right)^{-1} = \left(\sum_{i=1}^{\frac{99}{100}n}\frac{1}{\sigma^2} + \sum_{i=\frac{99n}{100}+1}^{n}\frac{1}{100\sigma^2}\right)^{-1} = \left(\frac{99}{100}\frac{n}{\sigma^2} + \frac{1}{10000}\frac{n}{\sigma^2}\right)^{-1}$$

$$\Rightarrow \sigma_{\mu_2}^2 = \left(\frac{9901}{10000}\frac{n}{\sigma^2}\right)^{-1} = \frac{10000}{9901}\frac{\sigma^2}{n} \Rightarrow \sigma_{\mu_2} = \frac{100}{\sqrt{9901}}\frac{\sigma}{\sqrt{n}} = \frac{100}{\sqrt{9901}}\sigma_\mu > \sigma_\mu$$

If we overweight 1% of the data by a factor of 100:

$$\sigma_{\mu_3}^2 = \left(\sum_{i=1}^{n}(1/\sigma_i^2)\right)^{-1} = \left(\sum_{i=1}^{\frac{99}{100}n}\frac{1}{\sigma^2} + \sum_{i=\frac{99n}{100}+1}^{n}\frac{100}{\sigma^2}\right)^{-1} = \left(\frac{99n}{100\sigma^2} + \frac{n}{\sigma^2}\right)^{-1}$$

$$\Rightarrow \sigma_{\mu_3}^2 = \left(\frac{199}{100}\frac{n}{\sigma^2}\right)^{-1} = \frac{100}{199}\frac{\sigma^2}{n} \Rightarrow \sigma_{\mu_3} = \sqrt{\frac{100}{199}}\frac{\sigma}{\sqrt{n}} = \sqrt{\frac{100}{199}}\sigma_\mu < \sigma_\mu$$

For the case where we underweighted 1% of the data by a factor of ~100, the error of the maximum likelihood became larger then the original σ_μ. For the case of overweighting the data the error was smaller than σ_μ.

One problem of weighting the data could be underestimating or overestimating the error bars on the maximum likelihood estimation. On the other hand, if we do not know exactly how the error is distributed on our data we will not know how to weight the data and could uncorrectly estimate the noise and the max. likelihood. If we have enough data and the fluctuations on the error are small we can just take the same σ and the estimation on σ_μ could be good enough, for example: $\frac{100}{\sqrt{9901}}\sigma_\mu \simeq 1.005\sigma_\mu$ and $\sqrt{\frac{100}{199}}\sigma_\mu \simeq 0.709\sigma_\mu$; we can see that underestimating the error, i.e., overweighting the data, the estimation of the error bars will be worse than the last case we analyzed. Therefore, overestimating the error (underweighting the data) for a small fraction of the data could not be too significant as $1.005\sigma_\mu - \sigma_\mu \simeq 0.005$. However, if the fluctuations are very notorious, weighting the data would be the best option, as we can see when $\frac{2}{\sqrt{3}}\sigma_\mu \simeq 1.155\sigma_\mu$

Problem 4

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
```

I defined the 51 points and the template Gaussian signal.

In [2]:

```
x=np.linspace(-5,5,51)
temp=np.exp(-(x**2)/2)
```

I defined the calculator routine to add a random Gaussian noise to the template Gaussian signal, so yn is the new signal that includes the noise. For this case the elements of the N matrix would be $N_{ij} = \sigma^2$ for $i = j$ and $N_{in} = 0$ for $i \neq j$, with σ the standard deviation of yn. For simplicity N^{-1} was defined just as a variable $N^{-1} = 1/\sigma^2$, so the value of $A^T N^{-1} A$ is just the dot product $(e^{-x^2/2})^T (e^{-x^2/2})/\sigma^2$. This is named as denom in the routine.

If yn=d then $A^T N^{-1} d$ becomes $\sum_i m_i d_i/\sigma^2$ with m_i the template model for each data point. This term is num in the routine and it will be a num for each data point. Finally, the estimated source amplitude is returned as num/denom, the signal-to-noise ratio is num/sqrt(denom) because the error is 1/sqrt(denom).

In [3]:

```
def calculator(x):
    xn=x
    temp=np.exp(-(xn**2)/2)
    noise=np.random.randn(51)
    yn=noise+temp
    noisec=np.std(yn)
    Ninv=1.0/(noisec**2)
    dat_filt=Ninv*yn
    denom=np.dot(temp,Ninv*temp)
    amp=[]
    snr=[]
    for i in range(len(xn)):
        tempp=np.exp(-((xn-xn[i])**2)/2)
        num=np.dot(tempp,dat_filt)
        amp.append(num/denom)
        snr.append(num/np.sqrt(denom))
    return np.array(amp), np.array(snr),1/np.sqrt(denom)
```

Below I used the calculator 100 times, so I have 100 amplitudes, 100 snr, and 100 errors.

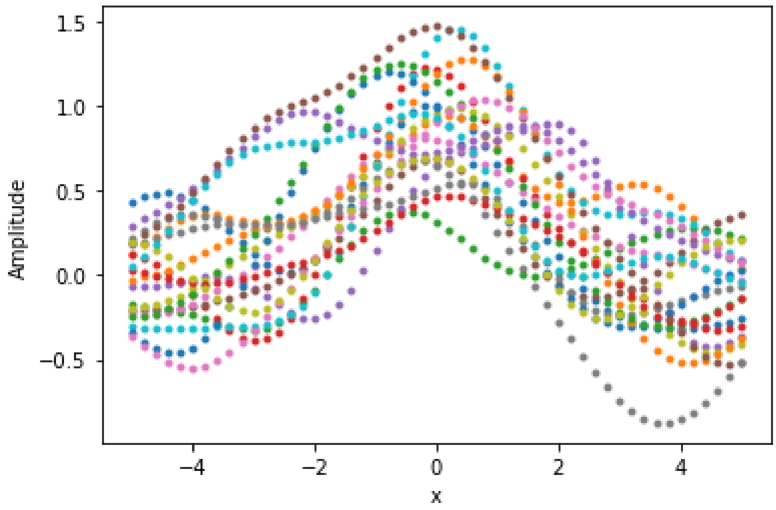
In [4]:

```
amp100=[]
snr100=[]
err100=[]
for i in range(100):
    amp,snr,err=calculator(x)
    amp100.append(amp)
    snr100.append(snr)
    err100.append(err)
```

In a plot for the amplitude for the first 20 models we can see that there is not a clear bias error on the signal.

In [5]:

```
for i in range(20):
    plt.plot(x,amp100[i],'.')
plt.xlabel('x')
plt.ylabel('Amplitude')
plt.show()
```



Now, let's take the weighted amplitude at the element x_j as $amp_{xj} = \sum_i amp_{ij}(err_i^{-2})/ \sum_i(err_i^{-2})$, and the i index goes from 1 to 100 since we used 100 models.

In [6]:

```
j100=[]
k100=[]
for i in range(len(amp100)):
    j100.append((amp100[i])*((err100[i])**-2))
    k100.append((err100[i])**-2)
```

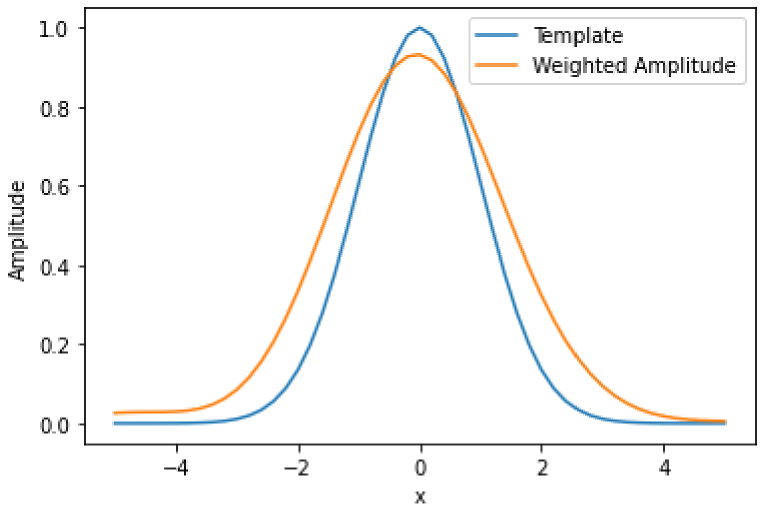
In [7]:

```
l100=[]
for i in range(len(snr100)):
    l100.append((snr100[i])*((err100[i])**-2))
```

A comparison between the weighted amplitude and the template signal.

In [8]:

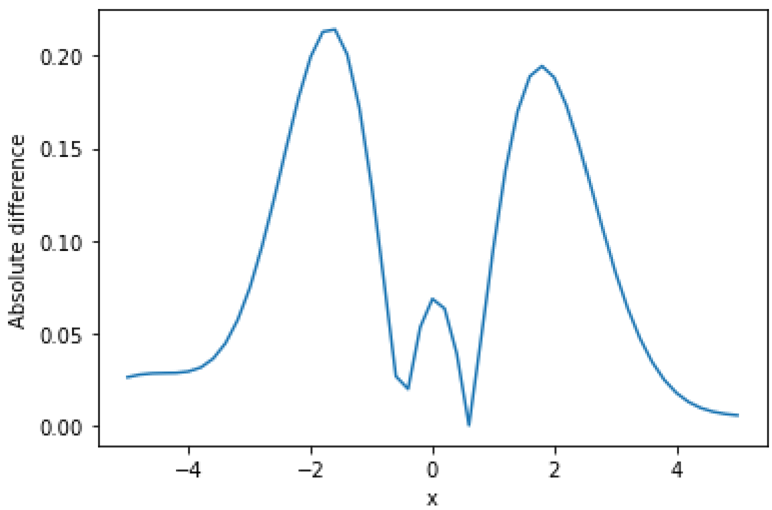
```
plt.plot(x,temp,label='Template')
plt.plot(x,sum(np.array(j100))/sum(k100),label='Weighted Amplitude')
plt.xlabel('x')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```



The absolute difference between the weighted amplitude and the template.

In [9]:

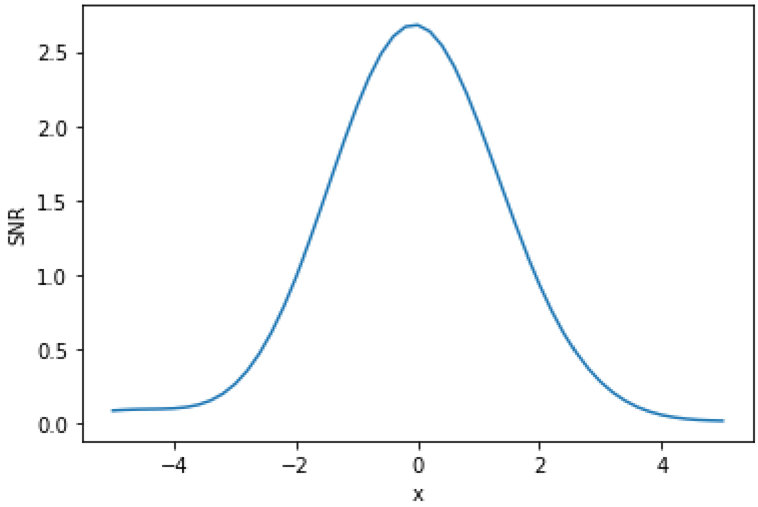
```
#plt.plot(x,temp)
plt.plot(x,np.abs(temp-sum(np.array(j100))/sum(k100)))
plt.xlabel('x')
plt.ylabel('Absolute difference')
plt.show()
```



It looks that the error is a bit biased but it could be just a uniform random difference if we stop analyzing here. I also add a SNR plot but does not give more information.

In [10]:

```
plt.plot(x,sum(np.array(l100))/sum(k100))
plt.xlabel('x')
plt.ylabel('SNR')
plt.show()
```



Let's try 10000 iterations

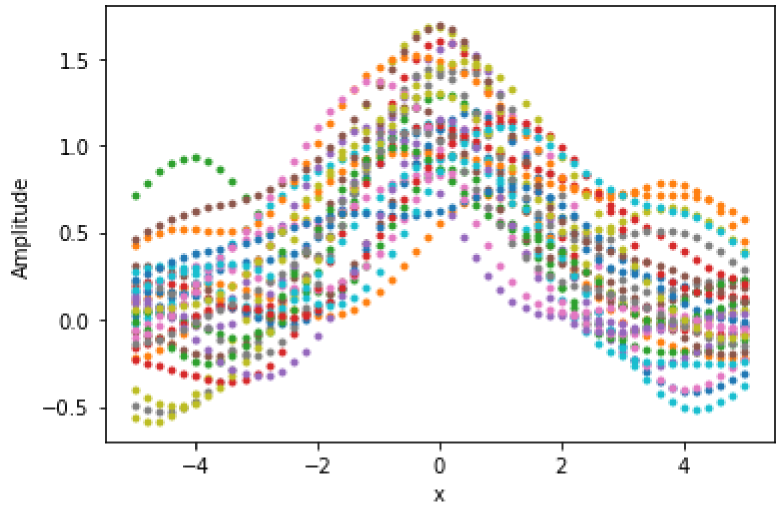
In [11]:

```
amp10000=[]
snr10000=[]
err10000=[]
for i in range(10000):
    amp,snr,err=calculator(x)
    amp10000.append(amp)
    snr10000.append(snr)
    err10000.append(err)
```

The amplitude for the first 30 models does not look biased too.

In [12]:

```
for i in range(30):
    plt.plot(x,amp10000[i],'.')
plt.xlabel('x')
plt.ylabel('Amplitude')
plt.show()
```



Let's obtain the weighted amplitude.

In [13]:

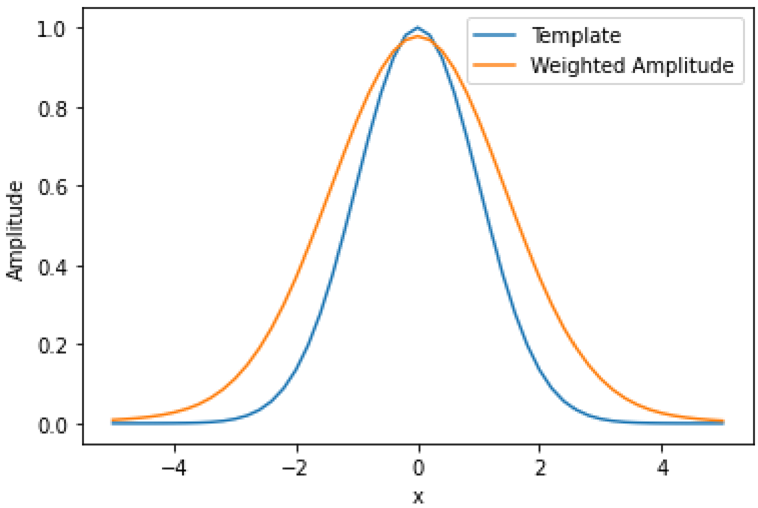
```
j10000=[]
k10000=[]
for i in range(len(amp10000)):
    j10000.append((amp10000[i])*((err10000[i])**-2))
    k10000.append((err10000[i])**-2)
```

In [14]:

```
l10000=[]
for i in range(len(snr10000)):
    l10000.append((snr10000[i])*((err10000[i])**-2))
```

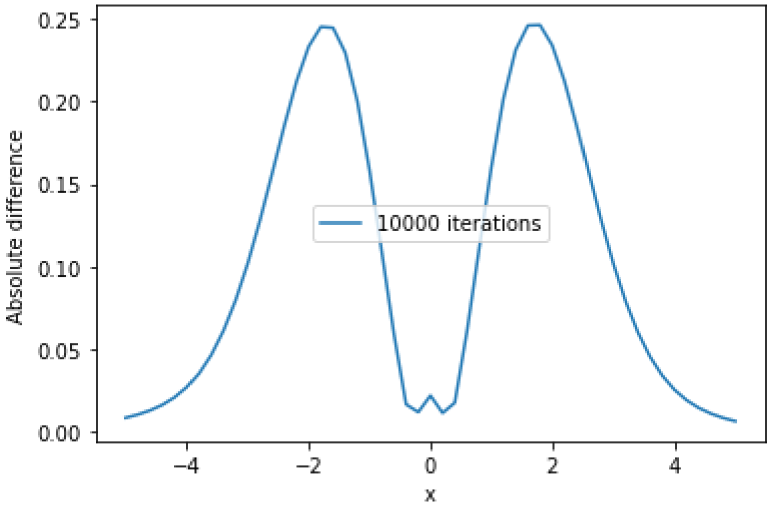
In [15]:

```
plt.plot(x,temp,label='Template')
plt.plot(x,sum(np.array(j10000))/sum(k10000),label='Weighted Amplitude')
plt.xlabel('x')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```



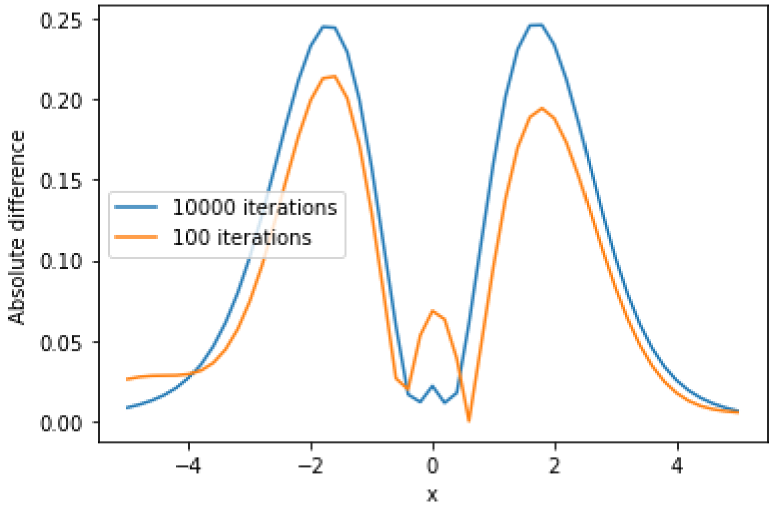
In [16]:

```
plt.plot(x,np.abs(temp-sum(np.array(j10000))/sum(k10000)),label='10000 iterations')
plt.xlabel('x')
plt.ylabel('Absolute difference')
plt.legend()
plt.show()
```



In [17]:

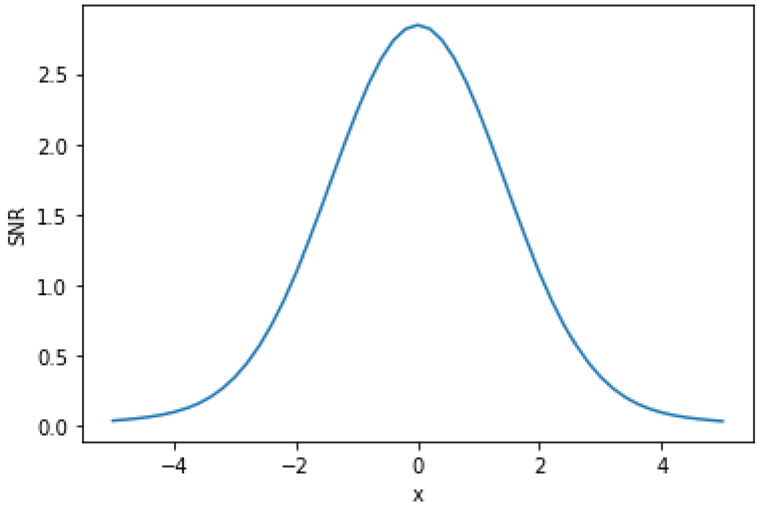
```
plt.plot(x,np.abs(temp-sum(np.array(j10000))/sum(k10000)),label='10000 iterations')
plt.plot(x,np.abs(temp-sum(np.array(j100))/sum(k100)),label='100 iterations')
plt.xlabel('x')
plt.ylabel('Absolute difference')
plt.legend()
plt.show()
```



Comparing the absolute differences for 10000 and 100 iterations, the 10000 iterations case looks more symmetrical with respect to x=0 and then is biased. Also the SNR is more symmetric.

In [18]:

```
plt.plot(x,sum(np.array(l10000))/sum(k10000))
plt.xlabel('x')
plt.ylabel('SNR')
plt.show()
```



Problem 5

If we add an invertible matrix S to our definition of the χ^2

$$\Rightarrow \chi^2 = (d - Am)^T N^{-1}(d - Am) = (d-Am)^T S^T (S^T)^{-1} N^{-1} S^{-1} S(d-Am)$$

$$= (S(d-Am))^T (SNS^T)^{-1}(S(d-Am)) = (Sd-SAm)^T (SNS^T)^{-1}(Sd-SAm)$$

Then, if we define $\tilde{d} = Sd$; $\hat{A} = SA$; $\tilde{N} = SNS^T$ as the new variables of the rotated space we can continue using the same definition for the χ^2:

$$\Rightarrow \chi^2 = (\tilde{d} - \tilde{A}m)^T \tilde{N}^{-1}(\tilde{d} - \hat{A}m)$$

The individual errors rotated are: $\tilde{n}_i = \sum_K S_{iK} n_K$. Then:

$$\langle \tilde{n}_i \tilde{n}_j \rangle = \left\langle \sum_K S_{iK} n_K \sum_l S_{jl} n_l \right\rangle$$

} While $\tilde{n}_i \tilde{n}_j$ could be correlated and $\neq 0$ when $j \neq i$; for the no correlated space $n_K n_l = 0$ when $K \neq l$.

$$\Rightarrow \langle \tilde{n}_i \tilde{n}_j \rangle = \left\langle \sum_K S_{iK} S_{jK} n_K^2 \right\rangle = \sum_K S_{iK} S_{jK} \sigma_K^2$$

On the other hand for the $\tilde{N}$ matrix we have:

$$(SN)_{iK} = S_{iK}\sigma_K^2 \Rightarrow \tilde{N}_{ij} = \sum_K (SN)_{iK} S^T_{Kj} = \sum_K S_{iK} S_{jK} \sigma_K^2$$

↳ $S^T_{Kj} = S_{jK}$

Then, $\tilde{N}_{ij} = \langle \tilde{n}_i \tilde{n}_j \rangle$ and the definition of χ^2 will be true even for correlated noises, we just have to be able to obtain $\langle n_i n_j \rangle$ as the components of the noises matrix.

↳ This is the for using the χ^2 because we can know the data and the model that we want to optimize, but if we do not know the relation $\langle n_i n_j \rangle$ of the noises we won't be able to set up the χ^2 correctly.