In [1]:

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

Question 1

First Order Moments

$$\begin{aligned}E[X] &= E[\mu_X + \sigma_A A] \\ &= E[\mu_X] + E[\sigma_A A] \\ &= \mu_X + \sigma_A E[A] \\ &= \mu_X\end{aligned}$$

where we used the linearity properties of the expectation value operator and the fact that $E[A] = 0$.

$$\begin{aligned}E[Y] &= E[\mu_Y + \sigma_A A + \sigma_B B] \\ &= E[\mu_Y] + E[\sigma_A A] + E[\sigma_B B] \\ &= \mu_Y + \sigma_A E[A] + \sigma_B E[B] \\ &= \mu_Y\end{aligned}$$

Second Order Moments

$$\begin{aligned}E[X^2] &= E[\mu_X^2] + 2E[\mu_X \sigma_A A] + E[\sigma_A^2 A^2] \\ &= \mu_X^2 + 2\mu_X \sigma_A E[A] + \sigma_A^2 E[A^2]\end{aligned}$$

Now using the fact that $E[A] = 0$ and $1 = E[A^2] - E[A]^2 = E[A^2]$ (A is normally distributed with standard deviation 1) we have

$$\begin{aligned} E[X^2] &= \mu_X^2 + \sigma_A^2 \\ &= \mu_X^2 + \sigma_A^2 \end{aligned}$$

We also have

$$\begin{aligned} E[Y^2] &= E[\mu_Y^2 + \sigma_A^2 A^2 + \sigma_B^2 B^2 + 2\mu_Y \sigma_A A + 2\mu_y \sigma_B B + 2\sigma_A \sigma_B AB] \\ &= \mu_Y^2 + \sigma_A^2 E[A^2] + \sigma_B^2 E[B^2] + 2\mu_y \sigma_A E[A] + 2\mu_y \sigma_B E[B] + 2\sigma_A \sigma_b E[AB] \\ &= \mu_Y^2 + \sigma_A^2 + \sigma_B^2 \end{aligned}$$

where we have used the fact that $E[A^2] = E[B^2] = 1$, $E[A] = E[B] = 0$, and finally that $\mathrm{cov}[A, B] = E[(A - E[A])(B - E[B])] = E[AB] = 0$ since A and B are assumed indepedent (and thus uncorrelated). We also have

$$\begin{aligned} E[XY] &= E[(\mu_X + \sigma_A A)(\mu_y + \sigma_A A + \sigma_B B)] \\ &= E[\mu_X \mu_Y + \mu_X \sigma_A A + \mu_X \sigma_B B + \mu_Y \sigma_A A + \sigma_A^2 A^2 + \sigma_A \sigma_B AB] \\ &= \mu_X \mu_y + \sigma_A^2 \end{aligned}$$

where once again we have used the linearity properties of the expectation value operator and the fact that $E[A^2] = E[B^2] = 1$, $E[A] = E[B] = 0$, and $E[AB] = 0$.

Variances, Covariance, and Correlation Coefficient

Since we have now computed all the first and second order moments, we can compute the desired quantities

Variances

Variance of X

$$\begin{aligned} \sigma_X^2 &= E[X^2] - E[X]^2 \\ &= \mu_X^2 + \sigma_A^2 - \mu_X^2 \\ &= \sigma_A^2 \end{aligned}$$

Variance of Y

$$\begin{aligned} \sigma_Y^2 &= E[Y^2] - E[Y]^2 \\ &= \mu_Y^2 + \sigma_A^2 + \sigma_B^2 - \mu_Y^2 \\ &= \sigma_A^2 + \sigma_B^2 \end{aligned}$$

Covariance

$$
\begin{aligned}
\text{cov[X,Y]} &= E[(X-\mu_X)(Y-\mu_Y)] \\
&= E[XY] - \mu_Y E[X] - \mu_X E[Y] + \mu_X\mu_Y \\
&= E[XY] - \mu_X\mu_y \\
&= \sigma_A^2
\end{aligned}
$$

Correlation Coefficient

$$
\begin{aligned}
\rho_{XY} &= \frac{\text{cov}[X, Y]}{\sigma_X\sigma_Y} \\
&= \frac{\sigma_A^2}{\sigma_A\sqrt{\sigma_A^2+\sigma_B^2}} \\
&= \frac{1}{\sqrt{1+(\sigma_B/\sigma_A)^2}}
\end{aligned}
$$

Question 2

The function is written below.

In [2]:
```
def getSample(n, ux=1, uy=1, sA=1, sB=1):
    norm1 = np.random.randn(n)
    norm2 = np.random.randn(n)
    X = ux + sA*norm1
    Y = uy + sA*norm1+sB*norm2
    return X,Y
```

Question 3

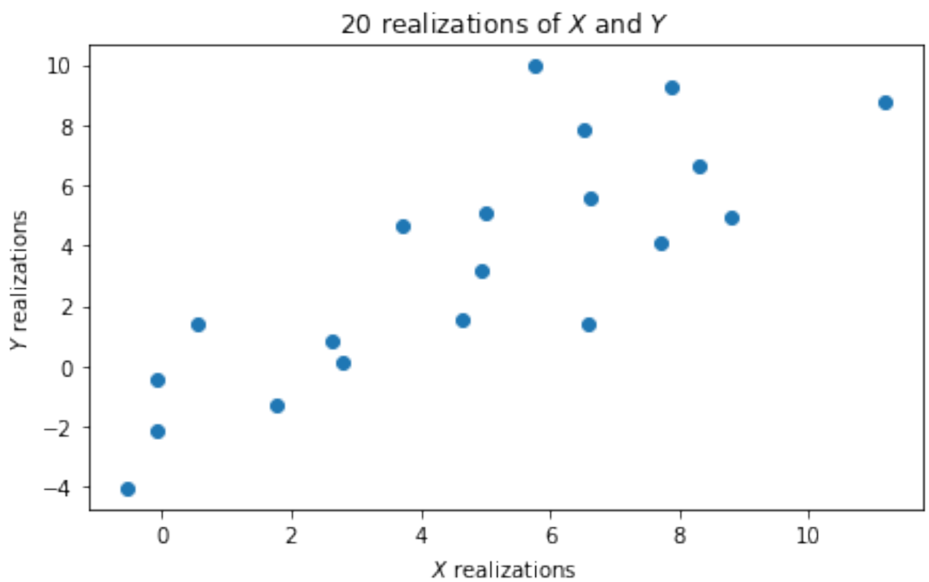
Define constants.

In [3]:
```
ux = 5
uy = 4
sA= 3
sB = 2
```

Get data.

In [4]:
```
x, y = getSample(20, ux=ux, uy=uy, sA=sA, sB=sB)
```

In [5]:
```
fig, ax = plt.subplots(figsize=(7,4))
ax.scatter(x,y)
ax.set_xlabel('$X$ realizations')
ax.set_ylabel('$Y$ realizations')
ax.set_title('20 realizations of $X$ and $Y$')
plt.show()
```



Each coefficient can be found explicitly or by using methods in the numpy package. We show both here.

Mean Values

From Sample

Explicitly...

In [6]:
```
ux_samp = np.sum(x)/len(x)
uy_samp = np.sum(y)/len(y)
print('Mean of X realizations: {}'.format(ux_samp))
print('Mean of Y realizations: {}'.format(uy_samp))
```

```
Mean of X realizations: 4.735117313106324
Mean of Y realizations: 3.3603048244348543
```

Using python functions...

In [7]:
```
ux_samp = np.mean(x)
uy_samp = np.mean(y)
print('Mean of X realizations: {}'.format(ux_samp))
print('Mean of Y realizations: {}'.format(uy_samp))
```

```
Mean of X realizations: 4.735117313106324
Mean of Y realizations: 3.3603048244348543
```

True Mean Value

In [8]:
```
print('True Mean X: {}'.format(ux))
print('True Mean Y: {}'.format(uy))
```

```
True Mean X: 5
True Mean Y: 4
```

Sample Variances and Covariance

Variance

Note that we are computing **biased** estimators for the variance here- as you have in your notes.

Sample Values

Explicitly:

In [9]:
```
Vx = np.sum((x-np.mean(x))**2)/len(x)
Vy = np.sum((y-np.mean(y))**2)/len(y)
print('Variance of X realizations: {}'.format(Vx))
print('Variance of Y realizations: {}'.format(Vy))
```

```
Variance of X realizations: 10.498718561004653
Variance of Y realizations: 14.992335300505465
```

Using numpy functionality:

In [10]:
```
Vx = np.var(x)
Vy = np.var(y)
print('Variance of X realizations: {}'.format(Vx))
print('Variance of Y realizations: {}'.format(Vy))
```

```
Variance of X realizations: 10.498718561004653
Variance of Y realizations: 14.992335300505465
```

True Values

In [11]:
```
Vx = sA**2
Vy = sA**2+sB**2
print('True variance of X: {}'.format(Vx))
print('True variance of Y: {}'.format(Vy))
```

```
True variance of X: 9
True variance of Y: 13
```

Covariance

Sample Values

Explicitly using the **biased** estimator (divide by $1/N$):

In [12]:
```
cov = np.sum((x-np.mean(x))*(y-np.mean(y)))/(len(x))
print('Sample covariance of X and Y: {}'.format(cov))
```

```
Sample covariance of X and Y: 10.264489740359114
```

Using numpy functionality. Note that numpy uses the **unbiased** estimator (divide by $1/(N-1)$):

In [13]:
```
cov = np.cov(x,y)[0,1]
print('Sample covariance of X and Y: {}'.format(cov))
```

```
Sample covariance of X and Y: 10.804726042483276
```

True Values

In [14]:
```
cov = sA**2
print('True covariance of X and Y: {}'.format(cov))
```

```
True covariance of X and Y: 9
```

Correlation Coefficient

Sample Values

Explicitly:

In [15]:
```
cov = np.sum((x-np.mean(x))*(y-np.mean(y)))/(len(x)-1)
Vx = np.sum((x-np.mean(x))**2)/(len(x)-1)
Vy = np.sum((y-np.mean(y))**2)/(len(y)-1)
corr_coef = cov/np.sqrt(Vx*Vy)
print('Sample correlation between X and Y: {}'.format(corr_coef))
```

```
Sample correlation between X and Y: 0.818153089695905
```

Using numpy funtionality:

In [16]:
```
corr_coef = np.corrcoef(x,y)[0,1]
print('Sample correlation between X and Y: {}'.format(corr_coef))
```

```
Sample correlation between X and Y: 0.818153089695905
```

True Values

In [17]:
```
corr_coef = 1/np.sqrt(1+(sB/sA)**2)
print('True Correlation between X and Y: {}'.format(corr_coef))
```

```
True Correlation between X and Y: 0.8320502943378437
```

Question 4

Define a function that computes the correlation coefficient given x and y

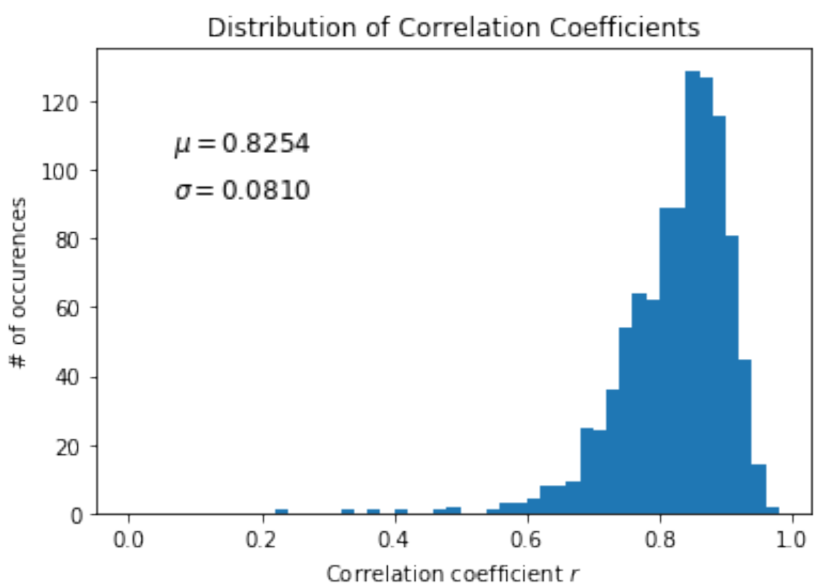
In [18]:
```
def get_corrcoef(x,y):
    return np.corrcoef(x,y)[0,1]
```

Compute many correlation coefficients for many samples. Here the sample size is 20.

In [19]:
```
corr_coeffs = np.array([get_corrcoef(*getSample(20, ux=5, uy=4, sA=3, sB=2)) \
                              for i in range(1000) ])
bins = np.arange(0, 1, 0.02)
cc_mean = np.mean(corr_coeffs)
cc_std = np.std(corr_coeffs)
```

Make a plot. The mean and standard deviation are shown on the plot.

In [20]:
```
plt.hist(corr_coeffs, bins=bins)
plt.ylabel('# of occurences')
plt.xlabel('Correlation coefficient $r$')
plt.text(0.05, 0.8, r'$\mu={:.4f}$'.format(cc_mean), fontsize=12, transform=ax.transAxes)
plt.text(0.05, 0.7, r'$\sigma={:.4f}$'.format(cc_std), fontsize=12, transform=ax.transAxes)
plt.title('Distribution of Correlation Coefficients')
plt.show()
```



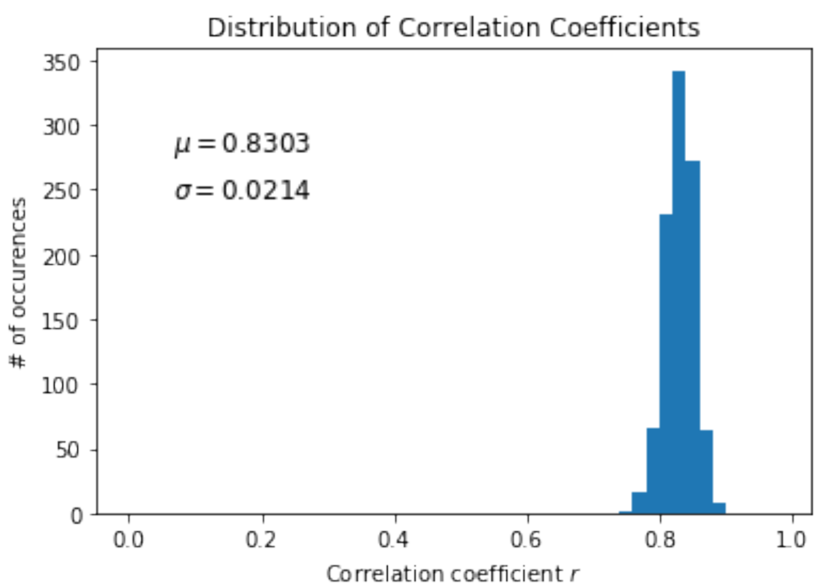
Question 5

Compute many correlation coefficients for many samples. Here the sample size is 200.

In [21]:
```
corr_coeffs = np.array([get_corrcoef(*getSample(200, ux=5, uy=4, sA=3, sB=2)) \
                        for i in range(1000) ])
bins = np.arange(0, 1, 0.02)
cc_mean = np.mean(corr_coeffs)
cc_std = np.std(corr_coeffs)
```

Make a plot.

In [22]:
```
plt.hist(corr_coeffs, bins=bins)
plt.ylabel('# of occurences')
plt.xlabel('Correlation coefficient $r$')
plt.text(0.05, 0.8, r'$\mu={:.4f}$'.format(cc_mean), fontsize=12, transform=ax.transAxes)
plt.text(0.05, 0.7, r'$\sigma={:.4f}$'.format(cc_std), fontsize=12, transform=ax.transAxes)
plt.title('Distribution of Correlation Coefficients')
plt.show()
```



A Neat Extension to These Problems

Lets create an animation to see how this changes in real time. I'll send you an email that includes the video.

For extra clarity, 10 times as many data points and bins were used for each plot in the animation. Since getSample is now called approxmately 1 million times, **the code takes approximately 2 minutes to run.**

Here we have a animation function, which is always used when creating animations in matplotlib. The parameter "i" specifies the frame number. In this case, the thing that changes is the number of data points created in the getSample function.

In [23]:
```
def animate(i):
    corr_coeffs = np.array([get_corrcoef(*getSample((i+1)*10, ux=5, uy
=4, sA=3, sB=2)) \
                            for j in range(10000) ])
    bins = np.arange(0, 1, 0.002)
    cc_mean = np.mean(corr_coeffs)
    cc_std = np.std(corr_coeffs)

    ax.clear()
    ax.hist(corr_coeffs, bins=bins)
    ax.set_ylabel('# of occurences')
    ax.set_xlabel('Correlation coefficient $r$')
    ax.text(0.05, 0.8, r'$\mu={:.4f}$'.format(cc_mean), fontsize=12, t
ransform=ax.transAxes)
    ax.text(0.05, 0.7, r'$\sigma={:.4f}$'.format(cc_std), fontsize=12,
transform=ax.transAxes)
    ax.set_title('Distribution of Correlation Coefficients $n=${}'.for
mat((i+1)*10))
```

Here we use the animation to generate a plot. When saving the plot, we specifify how many frames per second will be displayed in the mp4 file.

In [24]:
```
fig, ax = plt.subplots(1,1, figsize=(10,5))
ani = animation.FuncAnimation(fig,animate,80)
ani.save('testvid.mp4', fps=4)
plt.close()
```

Question 6

Here we take $N = 1000$ occurences of r for a sample size of $n = 20$.

In [25]:
```
n = 20
N=1000
```

Compute the correlation coefficients and then get an array of values of $\tanh^{-1}(r)$. These are distributed according to a Gaussian pdf. Also get the bins used in the histogram.

In [26]:
```
corr_coeffs = np.array([get_corrcoef(*getSample(n, ux=5, uy=4, sA=3, s
B=2)) \
                        for i in range(N) ])
z = np.arctanh(corr_coeffs)
z_mean = np.mean(z)
z_std = np.std(z)
bins = np.arange(0, 2, 0.1)
```

Obtain the theoretical Gaussian pdf with expectation value

$$E[Z] = \tanh^{-1}(\rho) + \frac{\rho}{2(n-1)}$$

and variance

$$V[Z] = \frac{1}{n-3}$$

which describes the distribution of the $\tanh^{-1}(r)$ values.

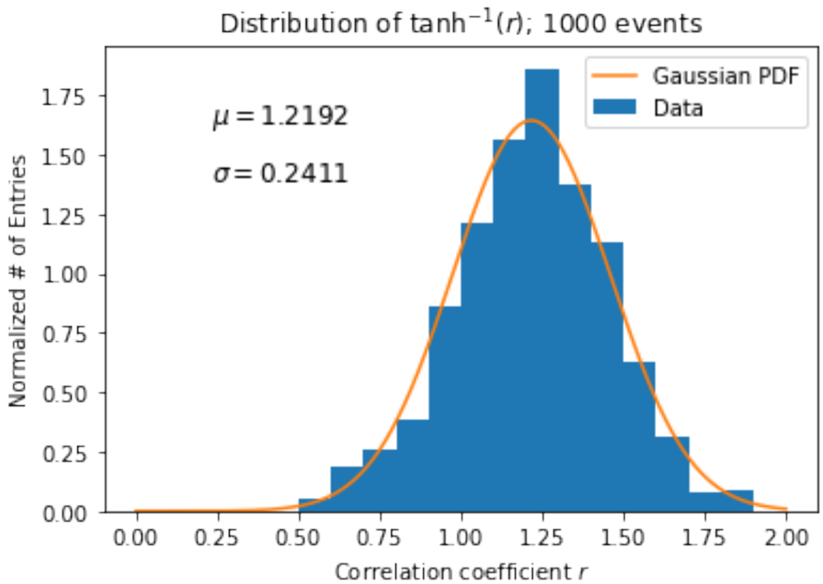
In [27]:
```
def gaussian_pdf(x, mu, sigma):
    return 1/np.sqrt(2*np.pi*sigma**2) * np.exp(-(x-mu)**2 / (2*sigma*
*2))

rho = 1/np.sqrt(1+(sB/sA)**2)
mu = np.arctanh(rho)+rho/(2*(n-1))
sigma = np.sqrt(1/(n-3))

x = np.linspace(0,2,1000)
y = gaussian_pdf(x,mu,sigma)
```

Plot.

In [28]:
```
plt.hist(z, bins=bins, density=True, label='Data')
plt.plot(x,y, label='Gaussian PDF')
plt.ylabel('Normalized # of Entries')
plt.xlabel('Correlation coefficient $r$')
plt.text(0.02, 0.65, r'$\mu={:.4f}$'.format(z_mean), fontsize=12, tran
sform=ax.transAxes)
plt.text(0.02, 0.55, r'$\sigma={:.4f}$'.format(z_std), fontsize=12, tr
ansform=ax.transAxes)
plt.title(r'Distribution of $\tanh^{-1}(r)$; 1000 events')
plt.legend()
plt.show()
```



In [29]:
```
print('Sample Mean: {}'.format(z_mean))
print('True Mean: {}'.format(mu))
```

```
Sample Mean: 1.2191775024199214
True Mean: 1.2166592776644212
```

In [30]:
```
print('Sample standard deviation: {}'.format(z_std))
print('True standard deviation: {}'.format(sigma))
```

```
Sample standard deviation: 0.24108693591886984
True standard deviation: 0.24253562503633297
```

It is worth noting that for finite sample sizes n, $z = \tanh^{-1}(r)$ is a biased estimator for $\tanh^{-1}(\rho)$ (which related to the true correlation coefficient). Note that this is taken into account in the plot above since we also include the $\frac{\rho}{2(n-1)}$ factor.