$$x_t = 0.7 * x_{t-1} - 0.1 * x_{t-2} + w_t$$

In [1]:
```
import numpy as np
from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
```

In [2]:
```
def acf_impl(x, nlags):
    """
    TODO
    @param x: a 1-d numpy array (data)
    @param nlags: an integer indicating how far back to compute the ACF
    @return a 1-d numpy array with (nlags+1) elements.
            Where the first element denotes the acf at lag = 0 (1.0 by d
efinition).
    """
    mean_x = x.mean()
    acfs = []

    acf_0 = 0
    for i in range(0, len(x)):
        acf_0 = acf_0 + ((x[i] - mean_x) * (x[i] - mean_x))
    acf_0 = acf_0 / len(x)

    for j in range(nlags+1):
        total = 0
        for i in range(0, len(x)-j):
            total = total + ((x[i+j] - mean_x) * (x[i] - mean_x))
        total = total / len(x)
        total = total / acf_0
        acfs.append(total)

    return acfs
```

In [3]:
```
# Analytical ACF calculations
phi_1 = 0.7
phi_2 = -0.1

rho_1 = phi_1 / (1 - phi_2)
rho_2 = ((phi_1 * phi_1)/(1 - phi_2)) + phi_2

print(f"Analytical -> rho_1 = {rho_1}, rho_2 = {rho_2}")

# Data simulation for empirical ACF
x_a, x_b = np.random.normal(0, 1), np.random.normal(0, 1) # Initially se
ries equal to white noise with variance 1

n = 10000
x = []
for i in range(n):
    x.append(0.7 * x_a - 0.1 * x_b + np.random.normal(0, 1))
    x_b = x_a
    x_a = x[-1]
x = np.array(x)
transient = 1000
x = x[transient::]

# Estimated ACF calculation
acf_val = acf_impl(x=x, nlags=2)

# Plots
print(f"Empirical -> rho_1 = {acf_val[1]}, rho_2 = {acf_val[2]}")
plt.figure()
plt.plot(acf_val, 'or', label='statsmodels empirical acf')
plt.plot([1, rho_1, rho_2], 'xb', label='own acf')
plt.legend();
plt.title('Empirical ACF vs Analytical ACF')
```

```
Analytical -> rho_1 = 0.6363636363636362, rho_2 = 0.34545454545454535
Empirical -> rho_1 = 0.6291197014298906, rho_2 = 0.33906375999370325
```

Out[3]:
```
Text(0.5, 1.0, 'Empirical ACF vs Analytical ACF')
```