CS 237 Fall 2019 Homework Ten Solution

Due date: PDF file due Friday November 22st @ 11:59PM in GradeScope with 12-hour grace period

No late submissions accepted

General Instructions

Please complete this notebook by filling in solutions where indicated. Be sure to "Run All" from the Cell menu before submitting.

There are two sections to the homework: problems 1 - 4 are analytical problems about last week's material, and the remaining problems are coding problems which will be discussed in lab next week.

In [2]:
```
# General useful imports
import numpy as np
from numpy import arange,linspace,mean, var, std, corrcoef, transpose, ones,log
from numpy.linalg import inv
from scipy.stats import pearsonr
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.mlab as mlab
from numpy.random import random, randint, uniform
import math
from collections import Counter
import pandas as pd
%matplotlib inline

# Basic Numpy statistical functions

X = [1,2,3]

# mean of a list
mean(X)             # might need to use np.mean, np.var, and np.std

# population variance
var(X)

# sample variance    ddof = delta degrees of freedom, df = len(X) - ddof
var(X,ddof=1)

# population standard deviation
std(X)

# sample standard deviation
std(X,ddof=1)


# Scipy statistical functions

# Scipy Stats Library Functions, see:
# https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.statistics.html


# Calculate the correlation coefficient rho(X,Y)


def rho(X,Y):
    return corrcoef(X,Y)[0][1]

def R2(X,Y):
    return corrcoef(X,Y)[0][1] ** 2

# Utility functions


# Round to 2 decimal places
def round2(x):
    return round(float(x)+0.00000000001,2)

# Round to 4 decimal places
def round4(x):
    return round(float(x)+0.00000000001,4)

def round4List(X):
    return [round(float(x)+0.00000000001,4) for x in X]

def probToPercent(p):
    pc = p*100
    if round(pc) == pc:
        return str(round(pc)) + "%"
    else:
        return str(round(pc,2))+ "%"
```

Homework 10 General Instructions

This homework contains analytical problems and also programming problems with some commentary on interpreting the results. The goal is simply to exercise your understanding of joint random variables, covariance, and correlation. An interesting application of correlation to real-world problems is presented.

Problem One (Joint Random Variables, Covarance, and Correlation)

Suppose we flip a coin and roll a single die. Define the following RVs based on this trial:

X = # heads in the toss of the coin.

Y = # dots showing on the die

Z = X + Y = total number of dots and heads showing.

You must fill in the following code template, writing explicit code to calculate the relevant statistics in two cases:

- The JRV is presented as two ranges Rx and Ry and a Joint PDF (a 2D matrix of probabilities); see the examples at the bottom of the code cell for illustrations;
- A list of points in 2D, for example, { (1,2), (3,3), (-2,3) } which are equiprobable and represented by two lists of values X = [1,3,-2], Y = [2,3,3]

We will distinguish these by using a pdf = [] to indicate the second situation.

You MUST Python to do this, but you can check your work with the Excel spreadsheet posted on the class web site. Round your results to 4 decimal places.

In [3]:
```
# Various statistical measures based on RVs which may not be
# equiprobable, so have to include 2D matrix pdf for PDF

# See examples at bottom



def my_mean(X,pdfX):
    return sum([X[i]*pdfX[i] for i in range(len(X))])

def my_var(X,pdfX):
    mu = my_mean(X,pdfX)
    return sum([(X[i]-mu)**2 * pdfX[i] for i in range(len(X))])

def my_std(X,pdfX):
    return my_var(X,pdfX)**0.5

# There are two cases for calculating cov and rho: you have
# a 2D pdf such as shown in lecture, OR you have a list of
# points (which are equiprobable). Must do these differently.
# If pdf != [], then JRV is (X,Y) with 2D pdf (see examples at bottom)
# if pdf == [], then X and Y must be same length and
# represent a set of points (x1,y1) ....  and pdf gives
# equal probability to all points.
# For efficiency, do these separately.

def my_cov(X,Y,pdf=[]):
    if pdf == []:
        sm = 0
        for i in range(len(X)):
            sm += X[i]*Y[i]
        return sm/len(X) - mean(X)*mean(Y)
    else:
        pdfX = [sum(pdf[r]) for r in range(len(X))]
        pdfY = [sum([pdf[r][c] for r in range(len(X))]) for c in range(len(Y))]
        muX = my_mean(X,pdfX)
        muY = my_mean(Y,pdfY)
        sm = 0
        for r in range(len(X)):
            for c in range(len(Y)):
                sm += X[r]*Y[c]*pdf[r][c]
        return sm - muX * muY

def my_rho(X,Y,pdf=[]):
    if pdf == []:
        sm = 0
        for i in range(len(X)):
            sm += X[i]*Y[i]
        return (sm/len(X) - mean(X)*mean(Y))/(std(X)*std(Y))
    else:
        pdfX = [sum(pdf[r]) for r in range(len(X))]
        pdfY = [sum([pdf[r][c] for r in range(len(X))]) for c in range(len(Y))]
        muX = my_mean(X,pdfX)
        muY = my_mean(Y,pdfY)
        sm = 0
        for r in range(len(X)):
            for c in range(len(Y)):
                sm += X[r]*Y[c]*pdf[r][c]

        sigmaX = my_std(X,pdfX)
        sigmaY = my_std(Y,pdfY)

        return ((sm - muX * muY)/(sigmaX * sigmaY))



X = [0, 1]
Y = [1,2,3,4,5,6]
Y1 = [6,5,4,3,2,1]
Z = [1,2,3,4,5,6,7]
```

```
# These are joint PDFs

#  X (rows) * Y (columns)

Pa = [ [1/12,1/12,1/12,1/12,1/12,1/12],
       [1/12,1/12,1/12,1/12,1/12,1/12]    ]

#   X * Z

Pb = [ [1/12,1/12,1/12,1/12,1/12,1/12, 0],
       [0, 1/12,1/12,1/12,1/12,1/12,1/12] ]

#   Y * Z

Pc = [ [1/12,1/12,0,0,0,0,0],
      [0,1/12,1/12,0,0,0,0],
      [0,0,1/12,1/12,0,0,0],
      [0,0,0,1/12,1/12,0,0],
      [0,0,0,0,1/12,1/12,0],
      [0,0,0,0,0,1/12,1/12]
          ]
print("\n(A)")
print("my_mean(X)  = " + str(round4(my_mean(Y,[0.1,0.1,0.2,0.2,0.3,0.1]))))  # should be 3.8
print("my_var(X)  = " + str(round4(my_var(Y,[0.1,0.1,0.2,0.2,0.3,0.1]))))    # should be 2.16
print("my_std(X)  = " + str(round4(my_std(Y,[0.1,0.1,0.2,0.2,0.3,0.1]))))    # should be 1.4697
print("\n(B)")
print("my_rho(X,Y,Pa)  = " + str(round4(my_rho(X,Y,Pa))))
print("my_cov(X,Y,Pa)  = " + str(round4(my_cov(X,Y,Pa))))    # should be 0.0
print("\n(C)")
print("my_rho(Y,Y1)  = " + str(round4(my_rho(Y,Y1))))   # should be -1.0
print("my_cov(Y,Y)  = " + str(round4(my_cov(Y,Y))))    # should be 2.9167
print("\n(D)")
print("my_rho(X,Z,Pb)  = " + str(round4(my_rho(X,Z,Pb))))
print("my_cov(X,Z,Pb)  = " + str(round4(my_cov(X,Z,Pb))))
print("\n(E)")
print("my_rho(Y,Z,Pc)  = " + str(round4(my_rho(Y,Z,Pc))))
print("my_cov(Y,Z,Pc)  = " + str(round4(my_cov(Y,Z,Pc))))    # should be 2.9167
```

```
(A)
my_mean(X) = 3.8
my_var(X) = 2.16
my_std(X) = 1.4697

(B)
my_rho(X,Y,Pa) = 0.0
my_cov(X,Y,Pa) = 0.0

(C)
my_rho(Y,Y1) = -1.0
my_cov(Y,Y) = 2.9167

(D)
my_rho(X,Z,Pb) = 0.281
my_cov(X,Z,Pb) = 0.25

(E)
my_rho(Y,Z,Pc) = 0.9597
my_cov(Y,Z,Pc) = 2.9167
```

Problem Two (Joint Random Variables, Covarance, and Correlation)

For each of the following, calculate the Correlation Coefficient ρ of the JRV (X, Y).

(A) Fahrenheit vs Celsius

In [4]:
```
Xfahrenheit = [45.2, 47.1, 47.5, 49.6, 49.8, 52.0, 54.3, 58.6, 63.2, 64.1]
Ycelsius = [7.8752, 8.117, 9.2009, 9.3167, 8.4564, 11.4075, 13.9236, 15.0762, 17.4678, 18.4362]

# Print out rho
print("rho(F,C) = ", round4(my_rho(Xfahrenheit,Ycelsius)))
```

```
rho(F,C) =  0.9817
```

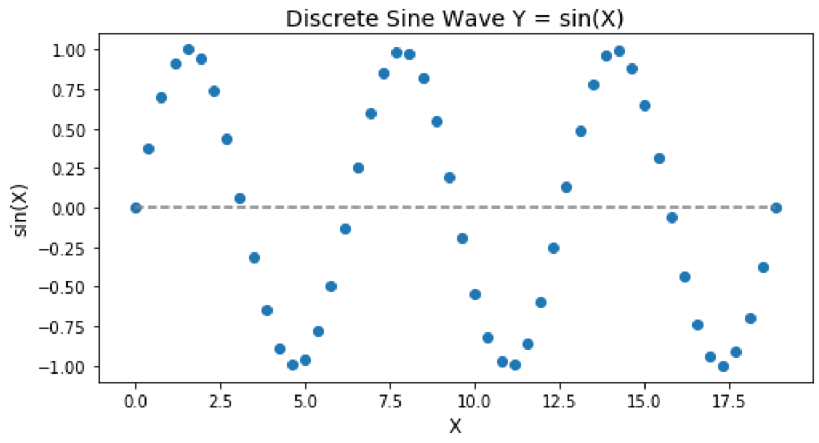
(B) Height and Weight for 2500 individuals

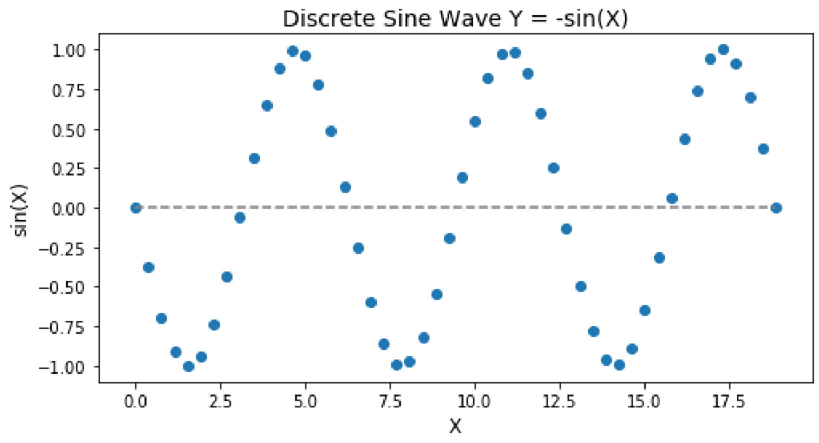
In [5]:
```
studs = pd.read_csv("http://www.cs.bu.edu/fac/snyder/cs237/Homeworks,%20Labs,%20and%20Code/biom
etricdata.csv")
X2b = list(studs['Height'])
Y2b = list(studs['Weight'])

# Print out rho
print("rho(X,Y) = ", round4(my_rho(X2b,Y2b)))
```

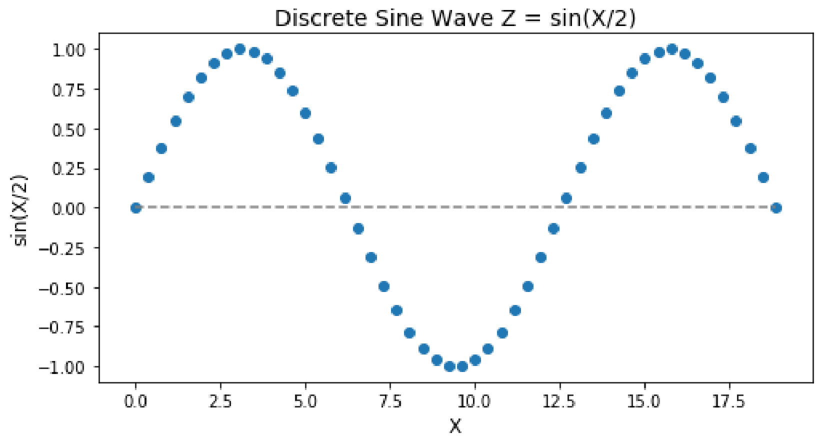
```
rho(X,Y) =  0.5029
```

(C) Discrete Sine Waves

In [6]:
```
X = linspace(0,6*math.pi,50)
Y = [math.sin(x) for x in X]
Yneg = [-y for y in Y]
Z = [math.sin(x/2) for x in X]

plt.figure(figsize=(8, 4))
plt.title('Discrete Sine Wave Y = sin(X)',fontsize=14)
plt.xlabel("X",fontsize=12)
plt.ylabel("sin(X)",fontsize=12)
plt.plot([0,X[-1]],[0,0],color="grey",linestyle="--")
plt.scatter(X,Y)

plt.figure(figsize=(8, 4))
plt.title('Discrete Sine Wave Y = -sin(X)',fontsize=14)
plt.xlabel("X",fontsize=12)
plt.ylabel("sin(X)",fontsize=12)
plt.plot([0,X[-1]],[0,0],color="grey",linestyle="--")
plt.scatter(X,Yneg)
plt.show()

plt.figure(figsize=(8, 4))
plt.title('Discrete Sine Wave Z = sin(X/2)',fontsize=14)
plt.xlabel("X",fontsize=12)
plt.ylabel("sin(X/2)",fontsize=12)
plt.plot([0,X[-1]],[0,0],color="grey",linestyle="--")
plt.scatter(X,Z)
plt.show()

# Print out rho
print("rho(Y,Y) = ", round4(my_rho(Y,Y)))

print("rho(Y,Yneg) = ", round4(my_rho(Y,Yneg)))

print("rho(Y,Z) = ", round4(my_rho(Y,Z)))
```







```
rho(Y,Y) =  1.0
rho(Y,Yneg) =  -1.0
rho(Y,Z) =  0.0
```

Problem Three (Joint Random Variables, Covarance, and Correlation)

Suppose there are 300 cards in a box numbered 1 through 300. Therefore, the number of each card has one, two, or three digits. A card is drawn at random from the box. Suppose that the number on the card has X digits of which Y are 0. Suppose we would like to know whether X and Y are correlated, negatively correlated, or uncorrelated. Determine $Cov(X, Y)$ and $\rho(X, Y)$, hence settling the question.

Solution:

Note that there are 9 one-digit numbers between 1 and 300, none of which is 0; there are 90 two-digit numbers of which 81 have no 0's, 9 have one 0, and none has two 0's; and there are 201 three-digit numbers of which 162 have no 0's, 36 have one 0, and 3 have two 0's. These facts show that as X increases so does Y. Therefore, X and Y are positively correlated. To be precise, let $f(x, y)$ be the joint PMF, which has the following matrix representation:

To see how we calculated the entries in the table, consider for example $f(3, 0)$. This quantity is $162/300$ because there are 162 three-digit numbers with no 0's. Now from this table we have that

$$\begin{aligned} E(X) &= 1 \cdot \frac{9}{300} + 2 \cdot \frac{90}{300} + 3 \cdot \frac{201}{300} = 2.64 \\ E(X^2) &= 1 \cdot \frac{9}{300} + 4 \cdot \frac{90}{300} + 9 \cdot \frac{201}{300} = 7.26 \\ E(Y) &= 0 \cdot \frac{252}{300} + 1 \cdot \frac{45}{300} + 2 \cdot \frac{3}{300} = 0.17 \\ E(Y^2) &= 0 \cdot \frac{252}{300} + 1 \cdot \frac{45}{300} + 4 \cdot \frac{3}{300} = 0.19 \\ E(XY) &= \sum_{x=1}^{3} \sum_{y=0}^{2} x \cdot y \cdot f(x, y) = 2 \cdot \frac{9}{300} + 3 \cdot \frac{36}{300} + 6 \cdot \frac{3}{300} = 0.48 \end{aligned}$$

Next, we have

$$\sigma_X = \sqrt{E(X^2) - E(X)^2} = \sqrt{7.26 - (2.64)^2} = \sqrt{0.2904}$$

and

$$\sigma_Y = \sqrt{E(Y^2) - E(Y)^2} = \sqrt{0.19 - (0.17)^2} = \sqrt{0.1611}$$

Therefore,

$$Cov(X, Y) = E(XY) - E(X) \cdot E(Y) = 0.48 - (2.64)(0.17) = 0.0312$$

and

$$\rho(XmY) = \frac{0.0312}{\sqrt{0.2904} \cdot \sqrt{0.1611}} = 0.1442,$$

which confirms that X and Y are positively correlated.

Problem Four (Correlation and Non-Linear Relationships)

(A) Let X be uniformly distributed over $[-1..1]$ and define $Y = X^2$. Show that $\rho(X, Y) = 0$. (This shows that RVs with a perfectly well-defined relationship may be uncorrelationed in terms of ρ, I showed this example in class but did not prove it.)

(B) Suppose we change the previous problem so that X is uniformly distributed over $[0..1]$ and again define $Y = X^2$. Calculate $\rho(X, Y)$ (it will not be 0).

(C) Show that if X and Y are independent, then $\rho(X, Y) = 0$. [Hint: first show that $E[X \cdot Y] = E[X] \cdot E[Y]$.]

Hint: For (b), note that X is continuous, so first solve:

$$E(X^n) = \int_0^1 x^n dx = ?$$

Solution:

(A)

$$Cov(X,Y) \;=\; E(X^3) - E(X)E(X^2) = 0$$

since $E(X) = 0$ and therefore $E(X^n) = E(X)^n = 0$ for $n > 0$. Thus $\rho(X,Y) = 0$ as well.

(B)

$$E(X^n) \;=\; \int_0^1 x^n dx \;=\; \left.\frac{x^{n+1}}{n+1}\right|_0^1 = \frac{1}{n+1}$$

thus $E(X) = 1/2$, $E(Y) = E(X^2) = 1/3$, and

$$\begin{aligned}\sigma_X^2 \;&=\; Var(X) = E(X^2) - E(X)^2 \;=\; 1/3 - (1/2)^2 = 1/12\\ \sigma_Y^2 \;&=\; Var(Y) = E(Y^2) - E(Y)^2 \;=\; 1/5 - (1/3)^2 = 4/45\end{aligned}$$

and finally

$$Cov(X,Y) \;=\; E(X^3) - E(X)E(X^2) \;=\; 1/4 - (1/2)(1/3) = 1/12$$

Therefore,

$$\rho(X,Y) \;=\; \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{1/12}{\sqrt{1/12}\cdot\sqrt{4/45}} = 0.9682.$$

(C) Since X and Y are independent, then for each $x \in R_X$ and each $y \in R_Y$ we have $f_{X,Y}(x,y) \;=\; f_X(x)\cdot f_Y(y)$. Then

$$\begin{aligned}E[X\cdot Y] &= \sum_{x\in R_X, y\in R_Y} x\cdot y\cdot f_{X,Y}(x,y)\\ &= \sum_{x\in R_X}\sum_{y\in R_Y} x\cdot y\cdot f_{X,Y}(x,y)\\ &= \sum_{x\in R_X}\sum_{y\in R_Y} x\cdot y\cdot f_X(x)\cdot f_Y(y)\\ &= \sum_{x\in R_X}\sum_{y\in R_Y}(x\cdot f_X(x))\cdot(y\cdot f_Y(y))\\ &= \sum_{x\in R_X}(x\cdot f_X(x))\cdot\sum_{y\in R_Y}(y\cdot f_Y(y))\\ &= \sum_{x\in R_X}(x\cdot f_X(x))\cdot E[Y]\\ &= E[Y]\cdot\sum_{x\in R_X}(x\cdot f_X(x))\\ &= E[Y]\cdot E[X]\\ &= E[X]\cdot E[Y]\end{aligned}$$

Therefore we have

$$\rho(X,Y) \;=\; \frac{E[X\cdot Y] - E[X]\cdot E[Y]}{\sigma_X\cdot\sigma_Y} = \frac{E[X]\cdot E[Y] - E[X]\cdot E[Y]}{\sigma_X\cdot\sigma_Y} = \frac{0}{\sigma_X\cdot\sigma_Y}.$$

Lab Problems: Correlation of Signals

This lab is about the use of correlation to determine the frequency of waveforms with a repetitive structure, such as musical signals. We will first understand what *signals* are, and then how correlation can be used to quantify how similar two signals are.

A **signal** X is just a function from non-negative integers into the reals:

$$X \,:\, [0, 1, . . , (N-1)] \;\rightarrow\; [-1..1]$$

where $t = 0, 1, 2, ... , N-1$ represents time (in some units, such as milliseconds) and $[-1..1]$ represents the amplitude of the signal. For instance here is an example adapted from Lab 9 which is just a sine wave, where $N = 315$:

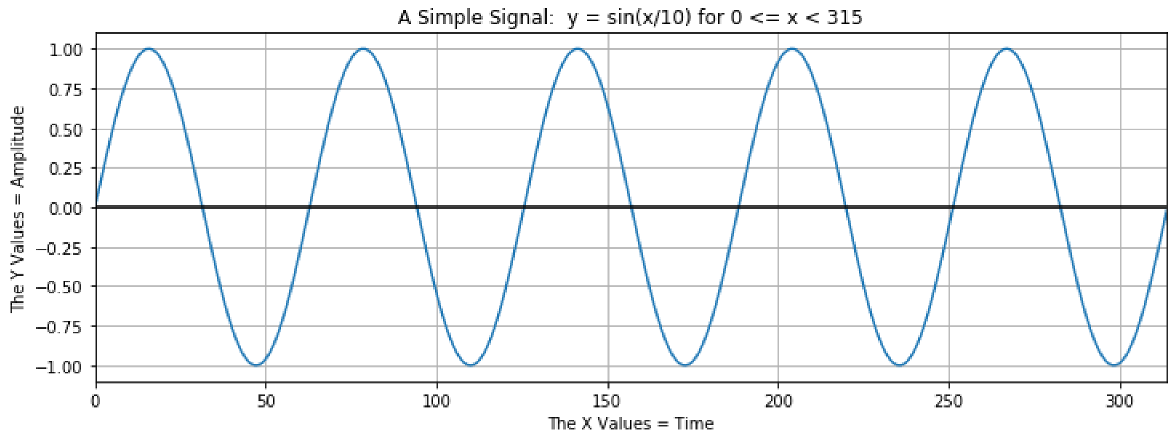
$$X(k) = sin(\,x/10\,)\ ,\ \text{for}\ k = 0, 1, \cdots, 314.$$

```
X =  [0.0885, 0.0932, 0.0936, 0.0915, 0.089, ..., -0.0009, -0.001, -0.001]
```

A **periodic signal** X has a pattern that repeats every P time units, where P is called the **period**. In this lab we will be dealing with signals that are periodic, or approximately periodic.

Plotting such a signal is easy, and note that we do not even need to give the x-axis, as it is assumed to be [0..len(X)-1].

In [7]:
```
N = 315
X = [np.sin(x/10) for x in range(N)]
plt.figure(figsize=(12,4))
plt.grid()
plt.plot(X)                    # when you don't give the x axis, it assumes it is [0,1, ..., len
(X)-1]
plt.plot([0,N-1],[0,0],color='black')
plt.xlim([0,N-1])
plt.title('A Simple Signal:  y = sin(x/10) for 0 <= x < 315')
plt.xlabel("The X Values = Time")
plt.ylabel("The Y Values = Amplitude")
plt.show()
```



For the rest of this lab, we will be dealing with sequences that are signals, but all you need to know is that they behave like random variables which, when you "poke" them, give the values $X[0], X[1], X[2], ...$ Therefore, it is not surprising that we can take the correlation of two signals.

The **correlation of two signals** of length N is exactly the same as if we consider them to be equiprobable, finite random variables:

$$Cov(X, Y) \;=\; E(X \cdot Y) - \mu_X \cdot \mu_Y \;=\; \frac{\sum_{i=0}^{N-1} X[i] \cdot Y[i]}{N} - \mu_X \cdot \mu_Y$$

$$\rho(X, Y) \;=\; \frac{Cov(X, Y)}{\sigma_X \cdot \sigma_Y}$$

Problem Five

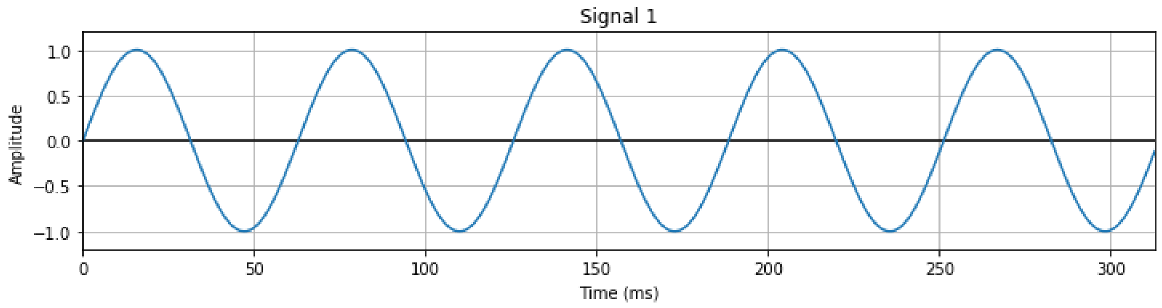
(A) Complete the following code template to calculate $\rho(X,Y)$, where we assume that X and Y are two signals of the same length. You may use `mean(...)` and `std(...)` from the numpy library, but otherwise write it yourself.

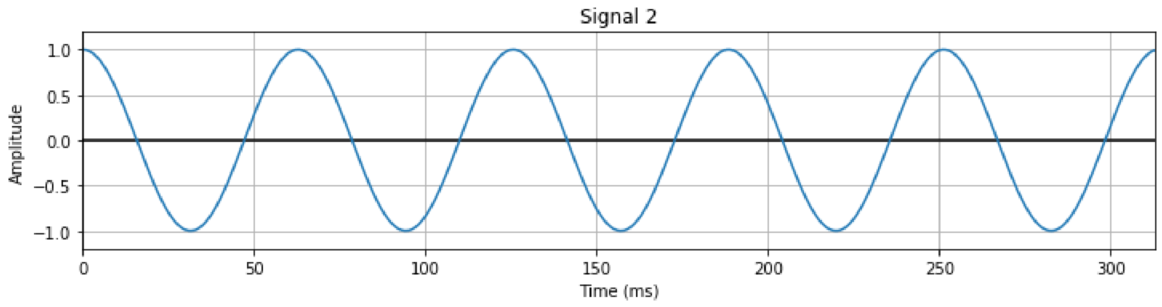
In [8]:
```
def rho(X,Y):
    mux = mean(X)
    muy = mean(Y)
    cov = 0
    for i in range(len(X)):
        cov += X[i]*Y[i]
    return (cov/len(X) - (mean(X) * mean(Y)))/(std(X)*std(Y))

# test

X = [1,2,3,4]
X1 = [1,1.1,1.2,1.3]
Y = [4,3,2,1]
Z = [2,4,3,5]

print(rho(X,X))    # should be 1.0
print(rho(X,Y))    # should be -1.0
print(rho(X,X1))   # should be 1.0
print(rho(X,Z))    # should be 0.8
```

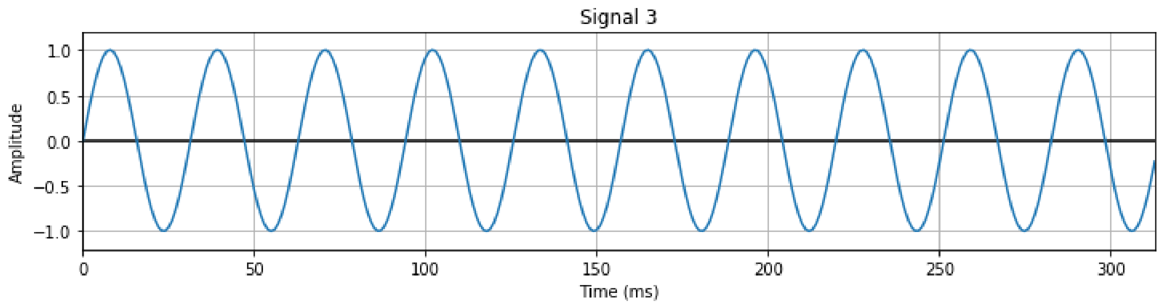
```
0.9999999999999998
-0.9999999999999998
1.0
0.7999999999999998
```

The following utility function simply displays three different signals lined up with respect to their x-axes.

In [9]:
```
# Just plotting three signals lined up along the x axis
def plot3Signals(X,Y,Z,title1="Signal 1", title2="Signal 2", title3="Signal 3"):
    N = len(X)
    fig = plt.figure(figsize=(12,10))
    fig.subplots_adjust(hspace=.5)
    ax1 = fig.add_subplot(311)

    plt.title(title1)
    plt.ylabel('Amplitude')
    plt.xlabel('Time (ms)')
    plt.plot([0,N-1],[0,0],color='black')
    plt.xlim([0,N-1])
    plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot(X)


    fig.add_subplot(312,sharex=ax1)

    plt.title(title2)
    plt.ylabel('Amplitude')
    plt.xlabel('Time (ms)')
    plt.plot([0,N-1],[0,0],color='black')
    plt.xlim([0,N-1])
    plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot(Y)


    fig.add_subplot(313,sharex=ax1)

    plt.title(title3)
    plt.ylabel('Amplitude')
    plt.xlabel('Time (ms)')
    plt.plot([0,N-1],[0,0],color='black')
    plt.xlim([0,N-1])
    plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot(Z)
    plt.show()

# example

T = list(range(314))
X = [math.sin(x/10) for x in T]
Y = [math.cos(x/10) for x in T]
Z = [math.sin(x/5) for x in T]

plot3Signals(X,Y,Z)
```







(B) Using the function in the previous code cell, print out the graphs for

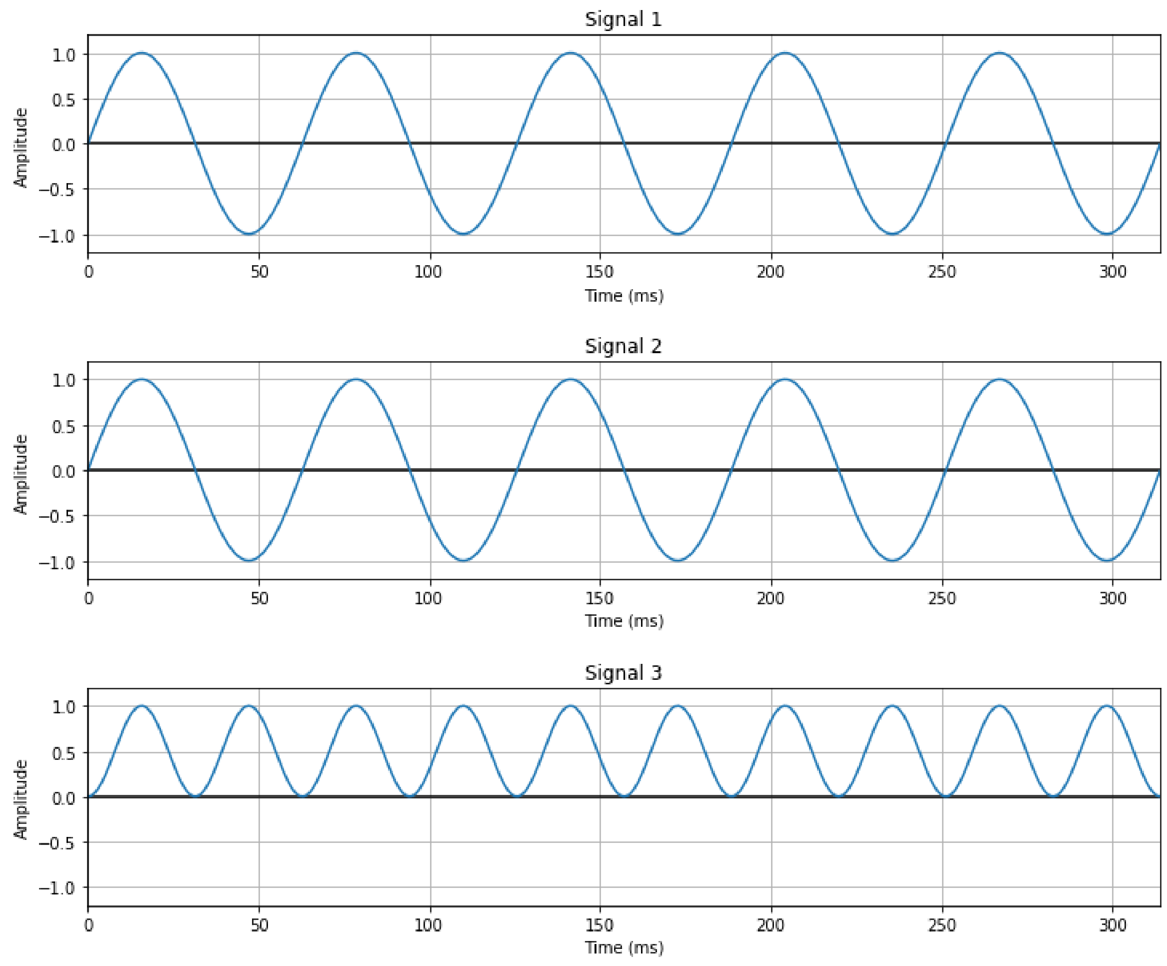
Signal 1: $X(k) = sin(x/10) \quad \text{for} \quad 0 \leq k < 315.$

Signal 2: $Y = X$

Signal 3: $Z = X \cdot Y$

and then print out $\rho(X, Y)$. This example was shown in lecture on the board.

In [10]:
```
# (B) solution

N = 315
X = [math.sin(x/10) for x in range(N)]
Y = X
Z = [X[i]*Y[i]  for i in range(N)]

plot3Signals(X,Y,Z)

print("rho(X,Y) = ", round4(rho(X,Y)))
```



```
rho(X,Y) =  1.0
```

(C) Now print out the graphs for these three collections of signals (which should be familiar to you from lecture):

First: Signal 1: $X(k) = sin(\, x/10\,) \quad \text{for} \quad 0 \leq k < 315.$

Signal 2: $Y = -X$

Signal 3: $Z = X \cdot Y$

and then print out $\rho(X, Y)$.

Second:

Signal 1: $X(k) = sin(\, x/10\,) \quad \text{for} \quad 0 \leq k < 315.$

Signal 2: $X(k) = cos(\, x/10\,) \quad \text{for} \quad 0 \leq k < 315.$

Signal 3: $Z = X \cdot Y$

and then print out $\rho(X, Y)$. (In this case, do not round to 4 decimal place, since it will be very close to 0.)

Third:

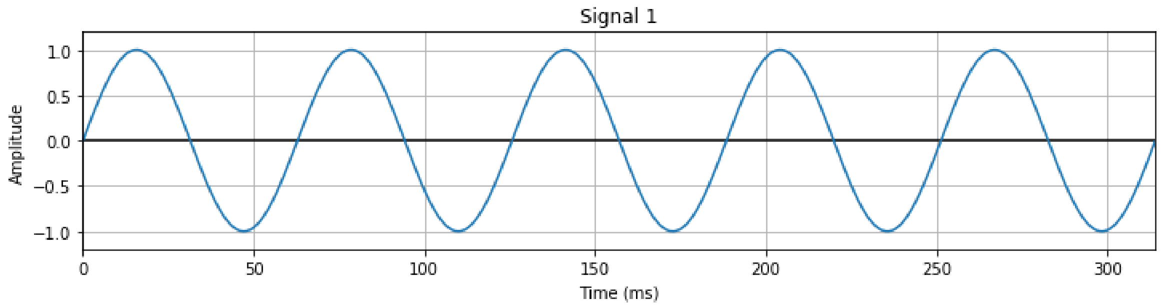
Signal 1: $X(k) = sin(\, x/10\,) \quad \text{for} \quad 0 \leq k < 315.$

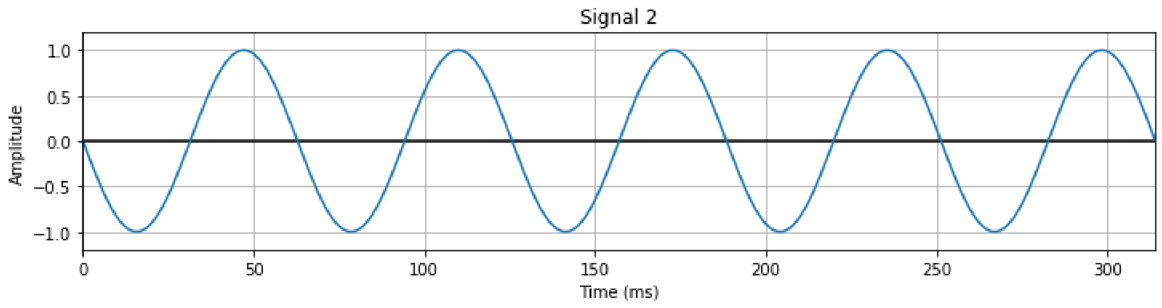
Signal 2: $X(k) = sin(\, x/5\,) \quad \text{for} \quad 0 \leq k < 315.$

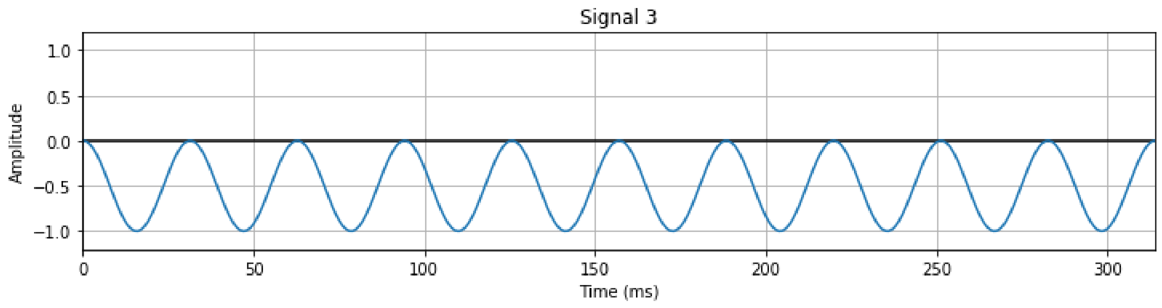
Signal 3: $Z = X \cdot Y$

and then print out $\rho(X, Y)$. (In this case, do not round to 4 decimal place, since it will be very close to 0.)

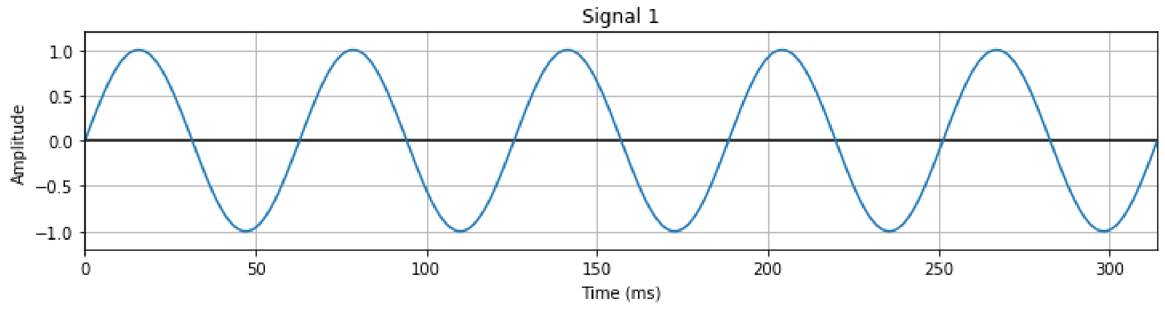
In [11]:
```
# (C) solution

N= 315
X = [math.sin(x/10) for x in range(N)]
Y = [-x for x in X]
Z = [X[i]*Y[i]  for i in range(N)]

plot3Signals(X,Y,Z)
print("rho(X,Y) = ", round4(rho(X,Y)))

X = [math.sin(x/10) for x in range(N)]
Y = [math.cos(x/10) for x in range(N)]
Z = [X[i]*Y[i]  for i in range(N)]

plot3Signals(X,Y,Z)
print("rho(X,Y) = ", rho(X,Y))

X = [math.sin(x/10) for x in range(N)]
Y = [math.sin(x/5) for x in range(N)]
Z = [X[i]*Y[i]  for i in range(N)]

plot3Signals(X,Y,Z)
print("rho(X,Y) = ", rho(X,Y))
```

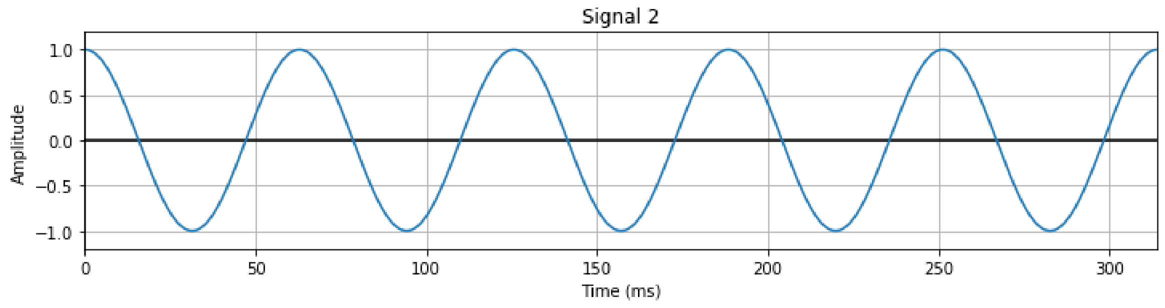

Signal 1
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300
Time (ms)

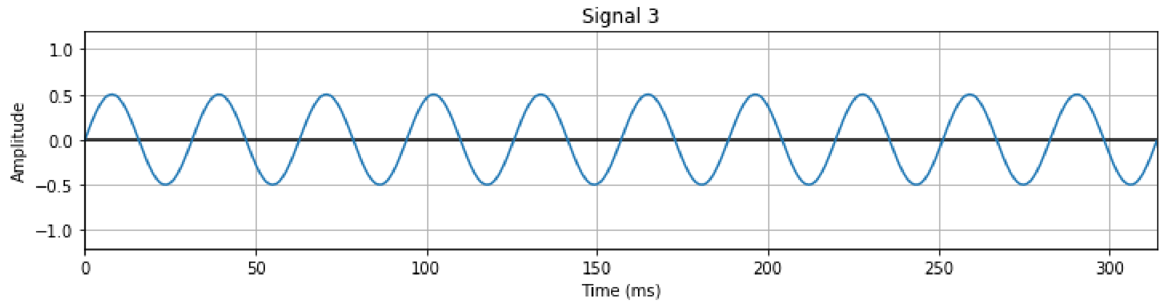

Signal 2
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300
Time (ms)


Signal 3
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0
50
100
150
200
250
300
Time (ms)

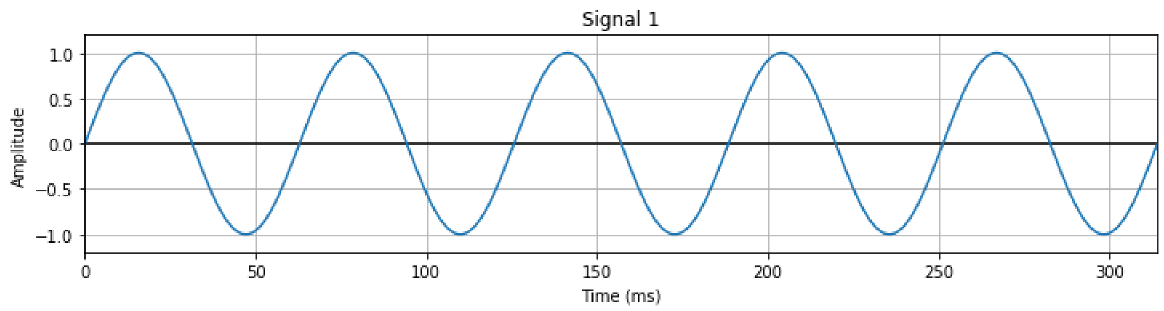
```
rho(X,Y) =  -1.0
```

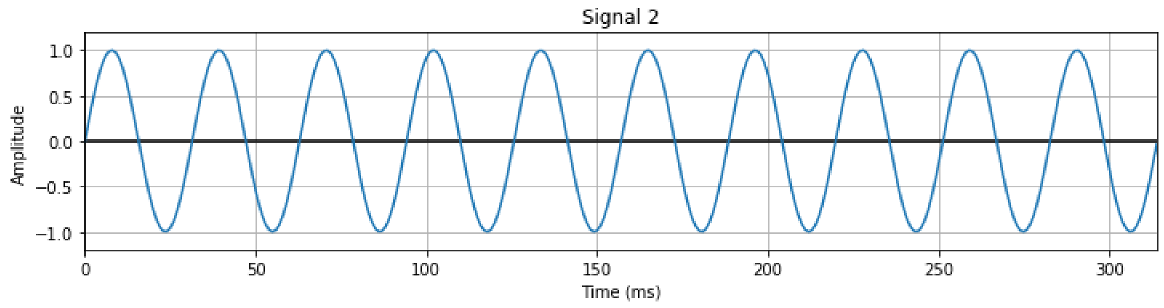


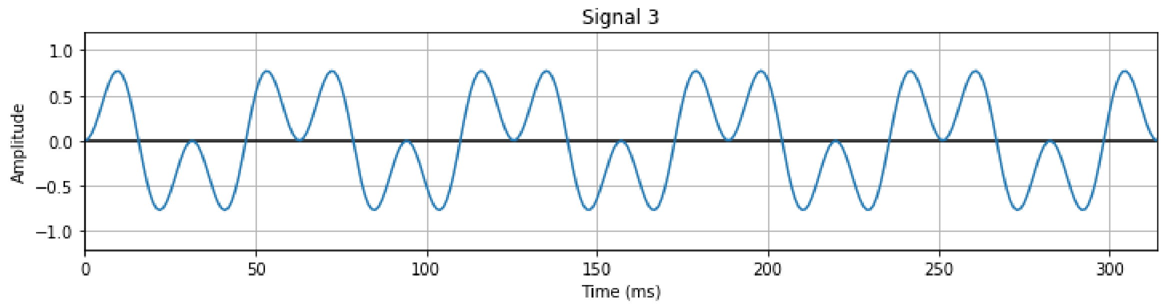




```
rho(X,Y) =  -4.2413870999628686e-05
```







```
rho(X,Y) =  -1.942708297111357e-06
```

(D) Your turn! Now I would like you to experiment with signals in the following form:

Signal 1: $X(k) = sin(\ x/10\ )$ for $0 \leq k < 315.$

Signal 2: $X(k) = sin(\ x/scale\ - lag)$ for $0 \leq k < 315.$

Signal 3: $Z = X \cdot Y$

where `scale` and `lag` have various values; scale will expand or contract X along the x-axis, and `lag` will shift it to the left or right along the x-axis.

Be sure to try setting `lag` to 0 and try various values of `scale` , and also setting `scale` to 10.0 and trying various values of `lag` .

In each case, examine the signals and also print out $\rho(X, Y)$, and then answer the following questions:

1. What happens when you fix `scale` at 10 and move `lag` ?
2. What happens when you fix `lag` at 0 and change `scale` ? Be sure to try integer values and also floats (e.g., 3.45, 11.83, etc.).
3. Supposing we set `scale` at 10 (as in the original) and vary `lag` . For which values of `lag` does $\rho(X, Y)$ most closely approach 1.0?

Solution:

1. In this case, the signal shifts back and forth along the x-axis, and ρ seems to vary quite a lot.
2. In this case, the signal expands and contracts, and it seems that ρ is close to 1 when scale is an integer, and close to 0 otherwise.
3. It seems to be at multiples of 63.

Problem Six (Correlation of Signals with Lags)

If we advance a signal, say by moving it `lag` time units to the left along the x-axis, then we have a *lagged signal*. If we have a formula for a signal, we can calculate this by simply subtracting the lag from the time before calculating the signal, as we did in the previous problem.

Sometimes this happens accidentally, say in YouTube, when the audio is out of sync with the video. In this problem we will do it deliberately, and investigate what happens to the correlation between a signal and a lagged version of itself.

Unfortunately, we are not always given a formula for a signal; sometimes we have a recording of an actual sound or some other real phenomenon, and we have to manipulate the original signal.

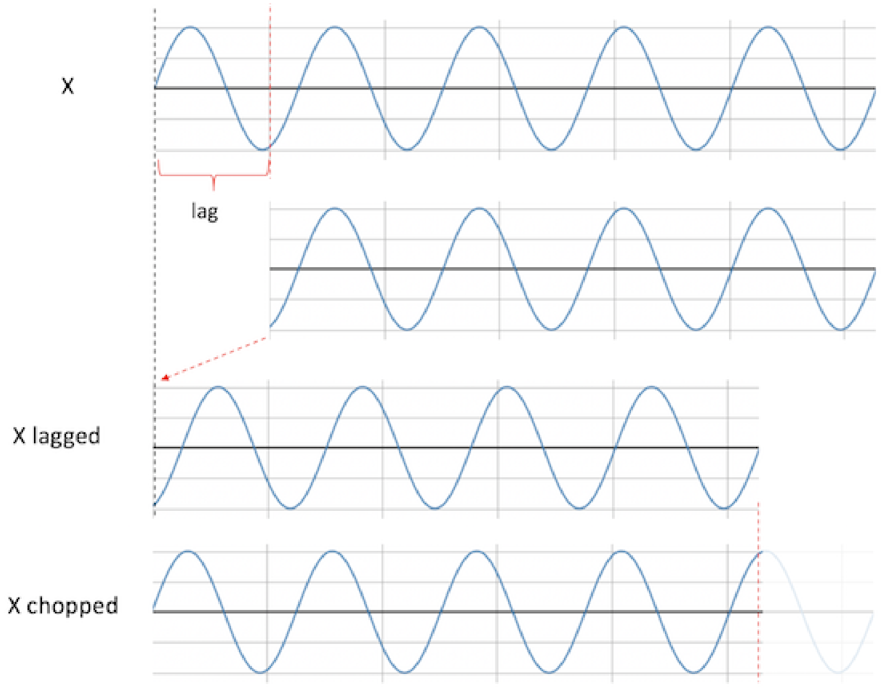
In order to compare such signals with lags, will do something a bit drastic: we will make a copy of the signal, and delete the first `lag` values from the front of the signal (thereby starting it at the value X[lag]:

```
Xlagged = X[lag:]
```

Now if we do this, then it is shorter than the original signal, so we will chop the same number of values from the end of X so we can compare them:

```
Xchopped = X[:-lag]
```

as illustrated here:



```
In [12]: # example

         X = list(range(10))
         print(X)
         lag = 3
         print(X[lag:])
         print(X[:-lag])
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6]
```

(A) Complete the following code template to take a time series, a signal, and a lag value, and return the chopped and lagged versions of the signal.

Hint: You need to be able to input a lag of 0, but this won't work with the Python code shown above, because X[:-0] will give you the empty list! So make a special case for the case of lag = 0.

In [13]:
```
# Return a pair (Xchopped, Xlagged)
def doLag(X,lag):
    if lag == 0:
        return (X,X)
    else:
        return (X[:-lag],X[lag:])

# test

X = list(range(10))
lag = 3
(X1,X2) = doLag(X,lag)
print(X)     # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(X1)    # [0, 1, 2, 3, 4, 5, 6]
print(X2)    # [3, 4, 5, 6, 7, 8, 9]
print()
X = list(range(10))
lag = 0
(X1,X2) = doLag(X,lag)
print(X)     # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(X1)    # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(X2)    # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6]
[3, 4, 5, 6, 7, 8, 9]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(B) Print out the graphs comparing X with a lagged version, where the lag = 10:

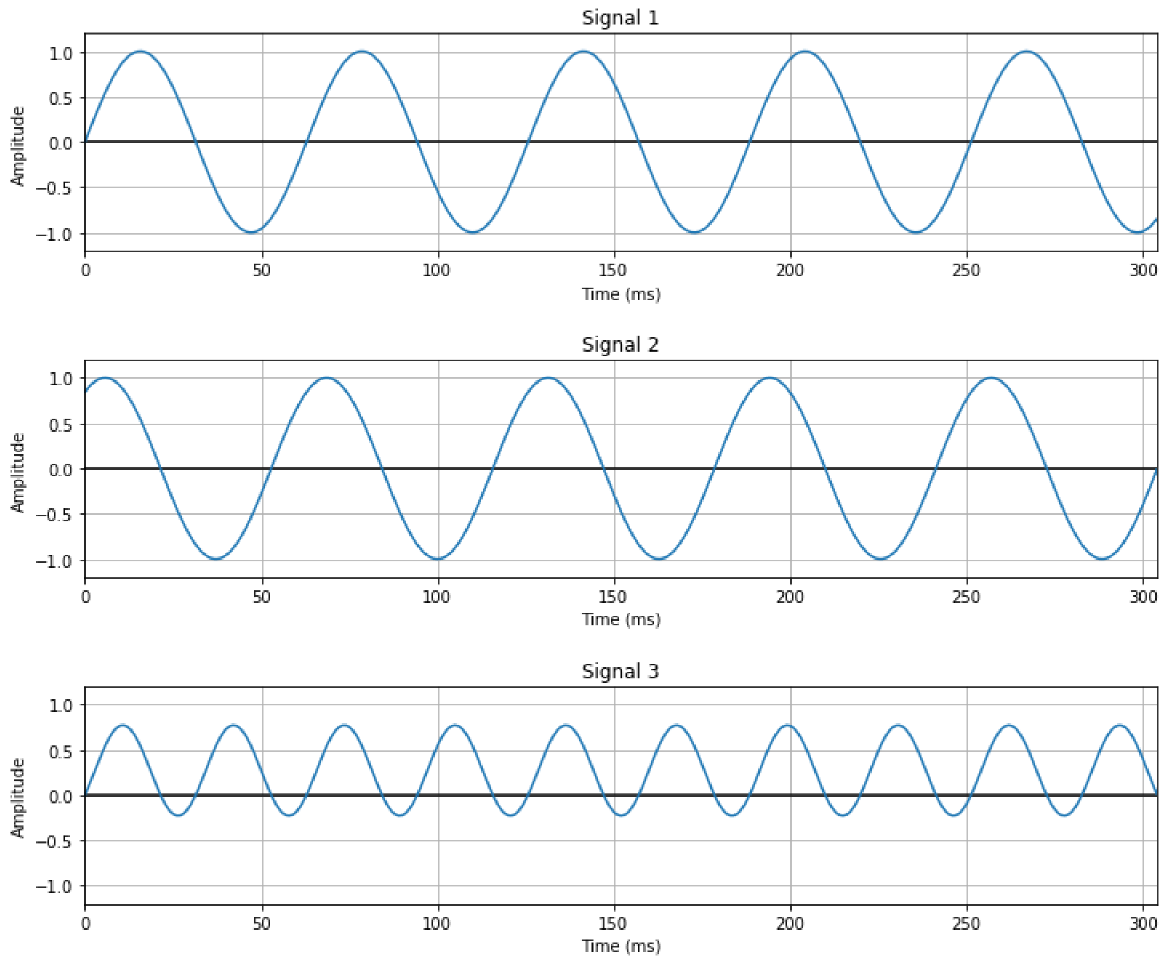
Signal 1: X' = X with 10 values chopped from the end

Signal 2: Y = X with 10 values chopped from the beginning

Signal 3: $Z = X' \cdot Y$

and then print out $\rho(X', Y)$.

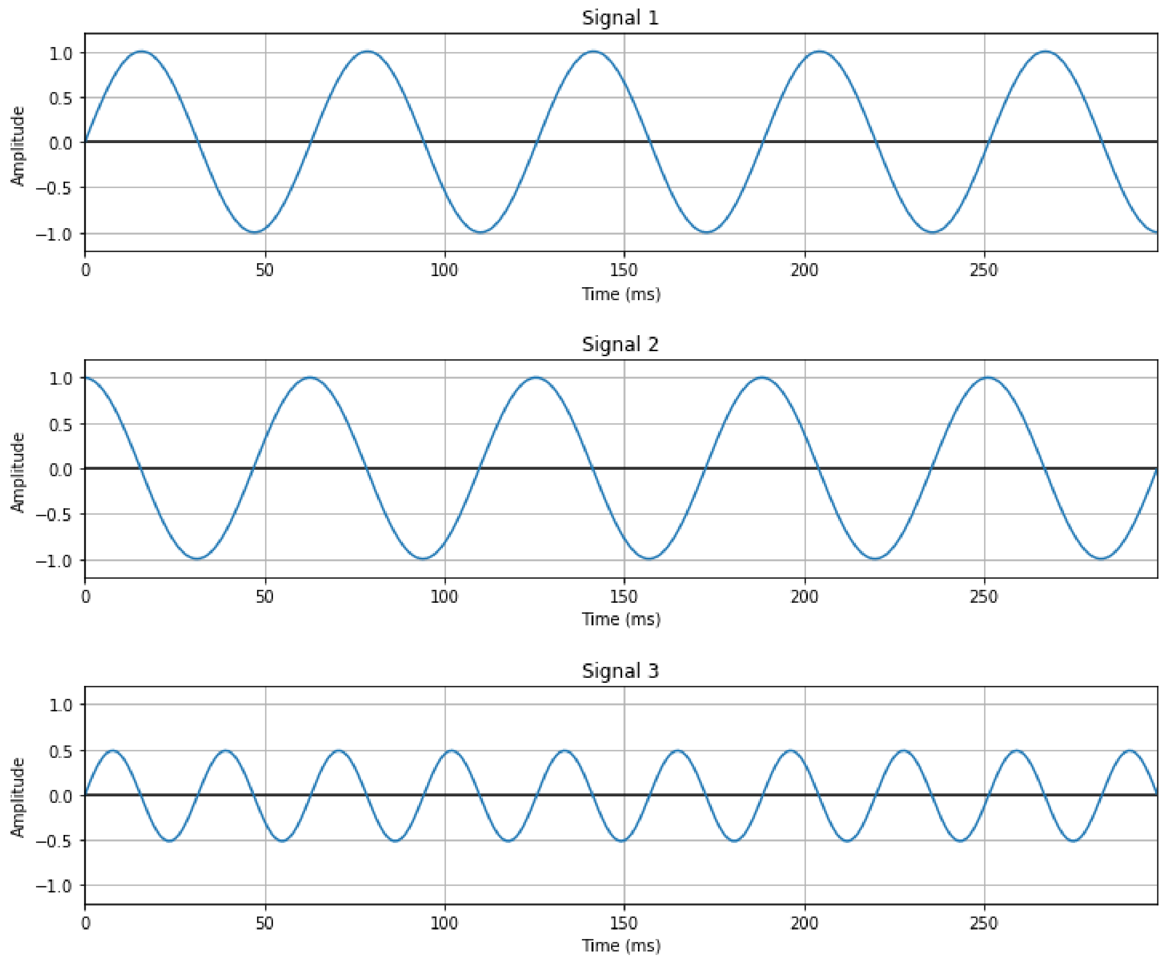
In [14]:
```
N = 315
X = [math.sin(x/10) for x in range(N)]
lag = 10
(X1,Y) = doLag(X,lag)
Z = [X1[t]*Y[t] for t in range(len(X1))]

plot3Signals(X1,Y,Z)

print("rho(X,Y) = ", rho(X1,Y))
```



```
rho(X,Y) =  0.5590821531104699
```

(C) Now do the same thing, but experiment and find a lag value that produces a ρ value as close to 0.0 as possible. You may ONLY use integer values for the lag, so you won't get exactly 0, but come as close as you can.

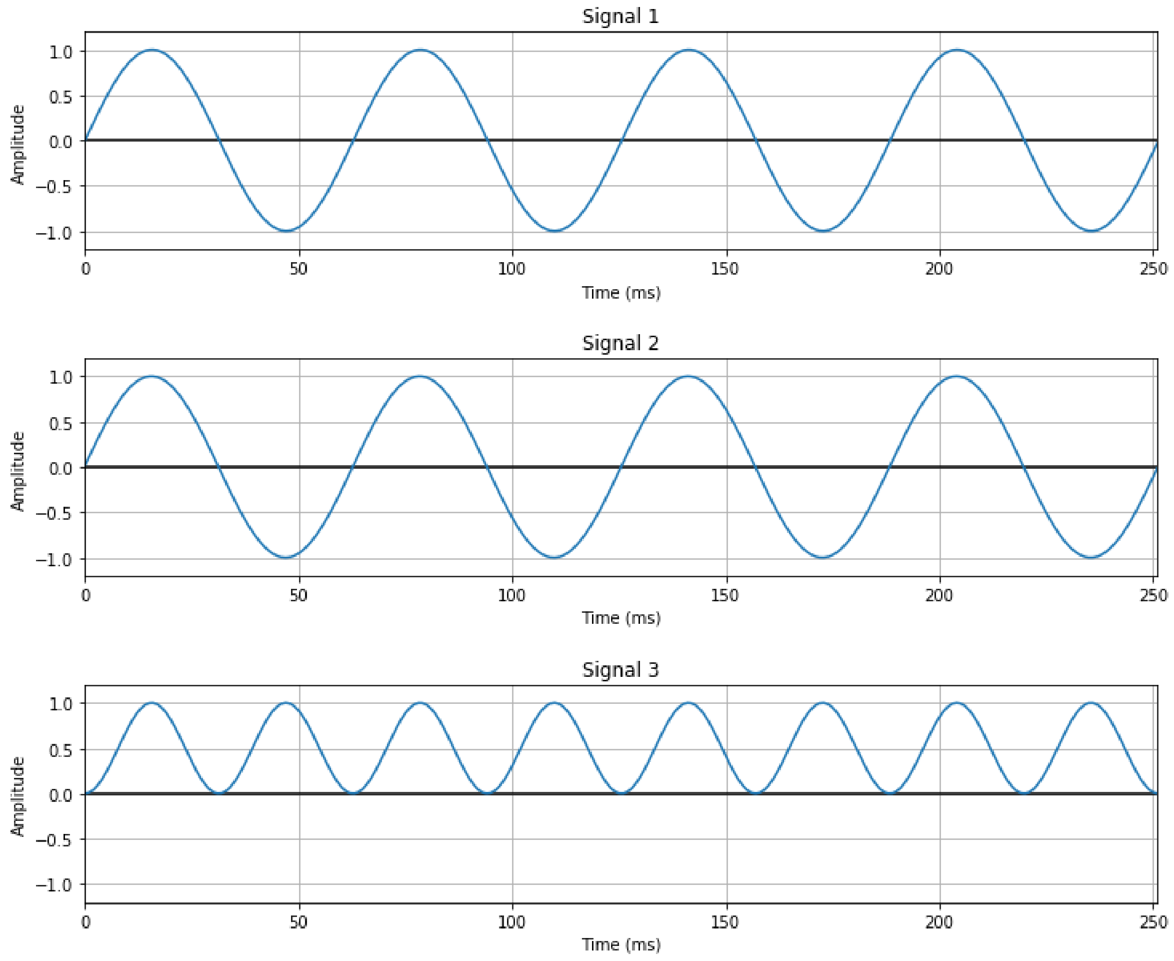
In [15]:
```
lag = 16
(X1,Y) = doLag(X,lag)
Z = [X1[t]*Y[t] for t in range(len(X1))]

plot3Signals(X1,Y,Z)

print("rho(X,Y) = ", rho(X1,Y))
```



```
rho(X,Y) =  0.006444480462264952
```

(D) Now do the same thing, but experiment and find a lag value that produces a ρ value as close to 1.0 as possible.

In [16]:
```python
lag = 63
(X1,Y) = doLag(X,lag)
Z = [X1[t]*Y[t] for t in range(len(X1))]

plot3Signals(X1,Y,Z)
print("rho(X,Y) = ", rho(X1,Y))
```



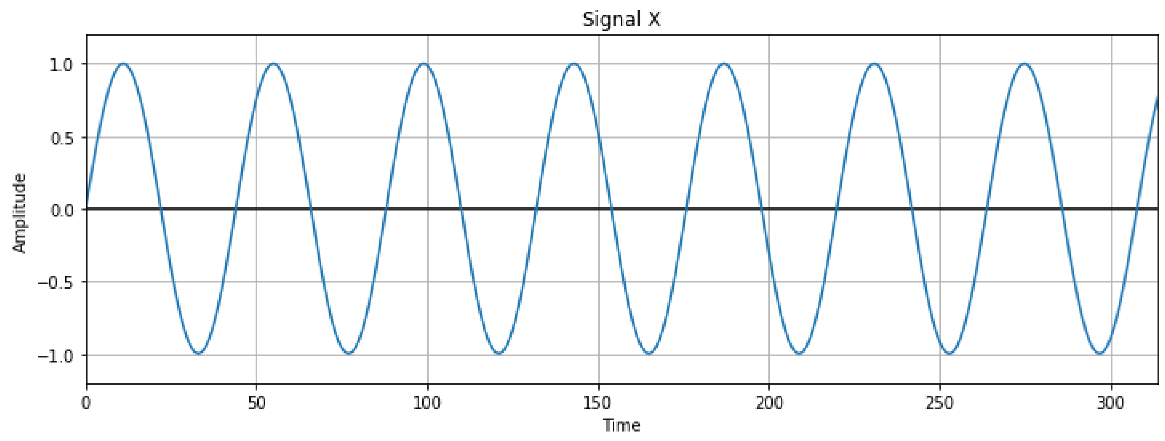
```
rho(X,Y) =  0.9998578812425687
```

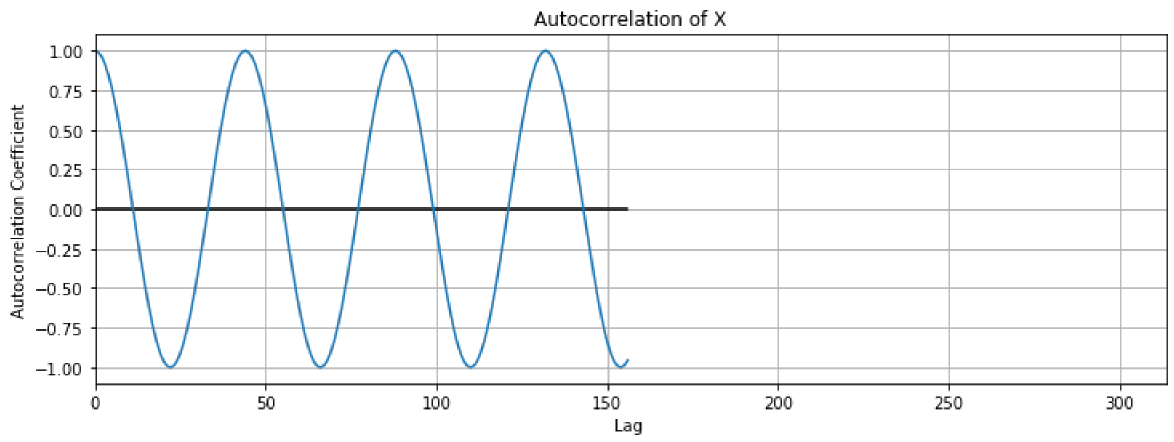
Problem Seven (Autocorrelation of Signals)

The *autocorrelation* of a signal is the correlation of the signal with a lagged version of itself. This autocorrelation can be used to determine whether the signal is periodic, i.e., repeats at fixed intervals, such as with a sine wave.

(A) Now we would like you to graph the correlation between a signal and a lagged version of itself, for lags from 0 to N/2. Do not draw the graphs for each signal, but just complete the template below and plot the ρ against the lag for the given signal X for lags = 0, 1, 2, ..., N/2.

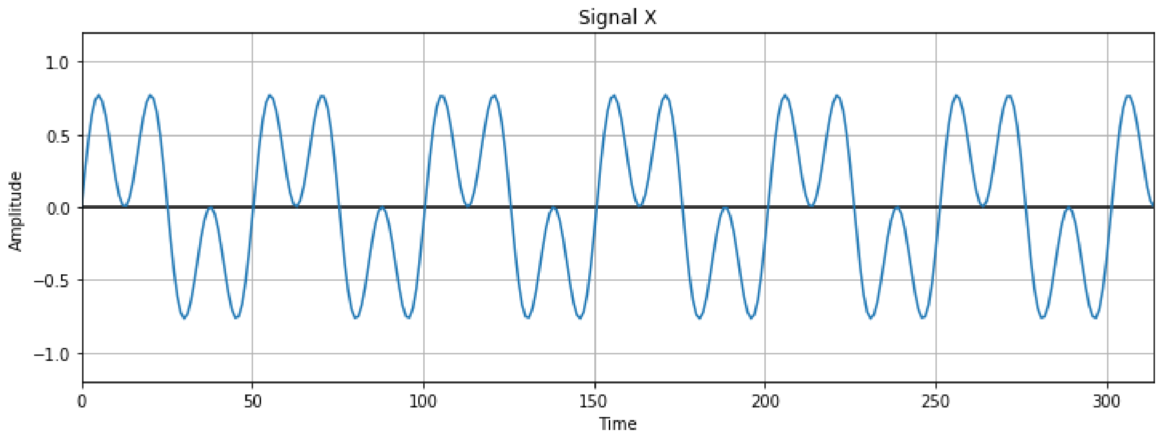
In [17]:
```
def showAutoCorrelation(X):
    N = len(X)
    fig = plt.figure(figsize=(12,10))
    fig.subplots_adjust(hspace=.5)
    ax1 = fig.add_subplot(211)

    plt.title("Signal X")
    plt.ylabel('Amplitude')
    plt.xlabel('Time')
    plt.xlim([0,N-1])
    plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot([0,N-1],[0,0],color='black')
    plt.plot(X)

    L = list(range(int(N/2)))
    M = len(L)
    A = [0]*(len(L))

    for lag in L:
        (X1,Y1) = doLag(X,lag)
        A[lag] = rho(X1,Y1)

    fig.add_subplot(212,sharex=ax1)

    plt.title("Autocorrelation of X")
    plt.ylabel('Autocorrelation Coefficient')
    plt.xlabel('Lag ')
    #plt.xlim([0,M-1])
    #plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot([0,M-1],[0,0],color='black')
    plt.plot(L,A)
    plt.show()


N = 315
X = [math.sin(x/7) for x in range(N)]


showAutoCorrelation(X)
```

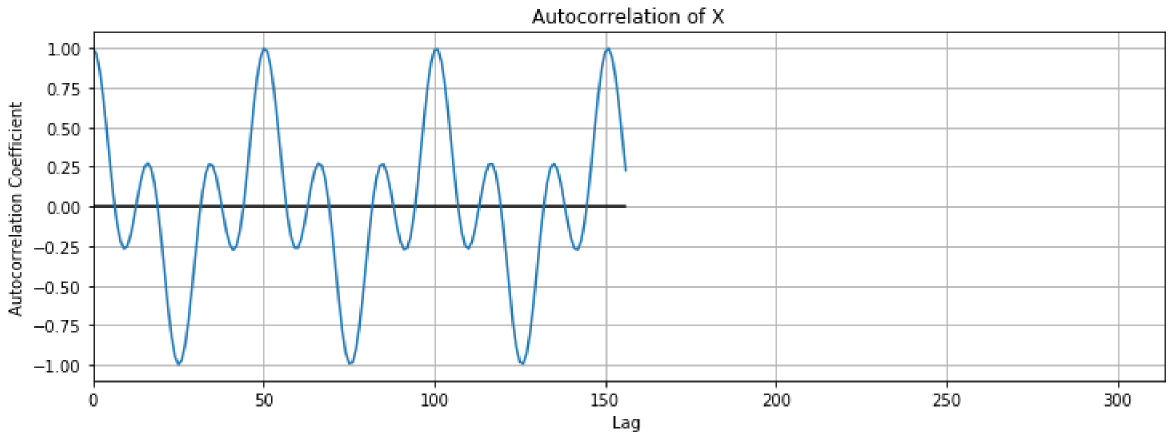




(B) Repeat for the following signal

In [18]:
```
N = 315
X = [math.sin(x/4)*math.cos(x/8) for x in range(N)]

showAutoCorrelation(X)
```





(C) The **period** of a periodic (i.e., repeating) signal is the time period between successive repeats of the pattern.

- What is the period (approximately) of the signal X in (B); and
- What do the maxima of the autocorrelation curve correspond to?

Solution:

- The period seems to be at 50, since at each lag of 50,100, 150, etc. the autocorrelation peaks at 1.0
- The peaks or maxima of the autocorrelation curve correspond to lags in at which the signal repeats.

Problem Eight (Fundamental Frequency and Peak Picking)

As we have just seen, the autocorrelation of a signal can be used to determine whether the signal is periodic, i.e., repeats at fixed intervals, such as with a sine wave.

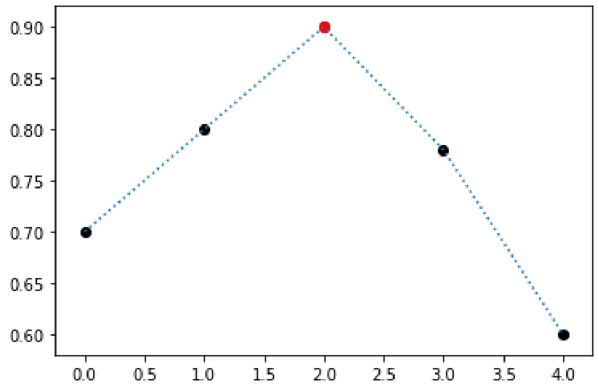
The **fundamental frequency** of a periodic signal is the smallest lag at which the signal repeats, which corresponds to the smallest lag at which there is a maximum in the autocorrelation curve. This represents the smallest possible period for the signal. Any other period will be a multiple of this shortest period, hence the name *fundamental*.

The maxima or **peaks** in any signal A are locations t at which

$$A[t-1] \;<\; A[t] \;>\; A[t+1]$$

as shown in red in the next figure:

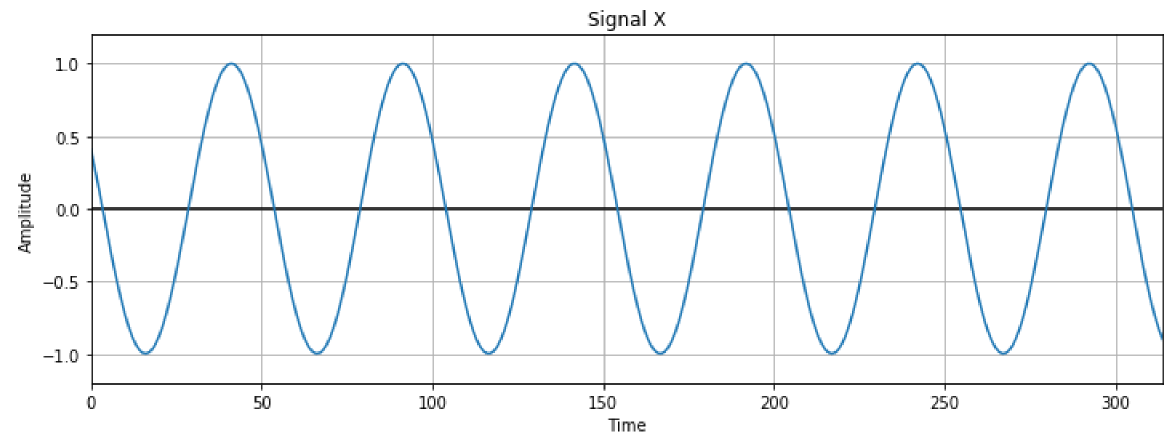
```
In [19]: Y=[0.7,0.8,0.9,0.78,0.6]
         X=[0,1,2,3,4]
         plt.plot(X,Y,ls=':')
         plt.scatter(X,Y,color='k')
         plt.scatter([2],[0.9],color='r')
         print()
```

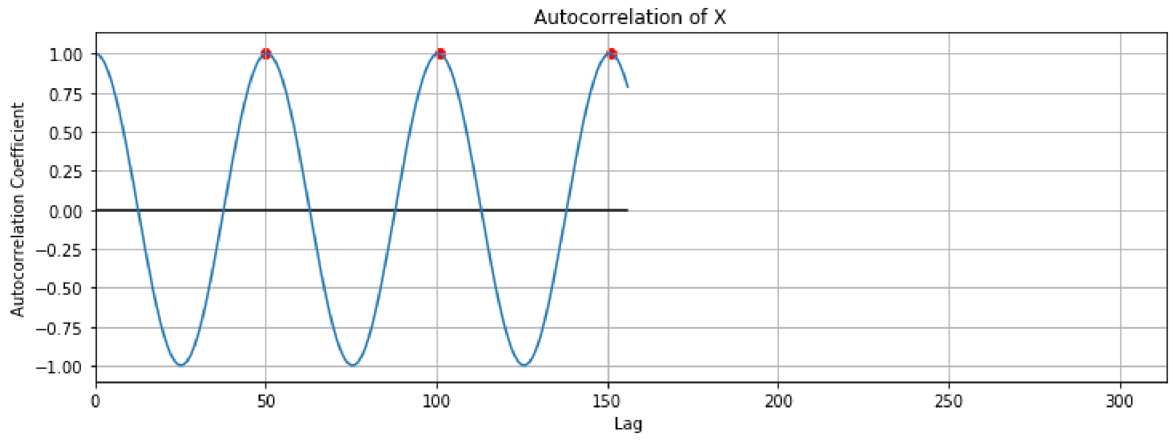


These peaks can be easily identified with a scan through the signal. Note that we are calling the autocorrelation curve A a *signal* because that is what it is!

(A) Rewrite the function from the previous problem, filling in the following template, and adding code to determine the peaks of the form $(t, A[t])$ (where A is the autocorrelation signal) and graph them in red together with the autocorrelation curve. Return the list peaks as the result of the function.

In [20]:
```
def showAutoCorrelationWithPeaks(X):
    N = len(X)
    fig = plt.figure(figsize=(12,10))
    fig.subplots_adjust(hspace=.5)
    ax1 = fig.add_subplot(211)

    plt.title("Signal X")
    plt.ylabel('Amplitude')
    plt.xlabel('Time')
    plt.xlim([0,N-1])
    plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot([0,N-1],[0,0],color='black')
    plt.plot(X)

    L = list(range(int(N/2)))
    M = len(L)
    A = [0]*(len(L))

    for lag in L:
        (X1,Y1) = doLag(X,lag)
        A[lag] = rho(X1,Y1)

    peaks = []
    for t in range(1,len(L)-1):
        if A[t-1] < A[t] and A[t] > A[t+1]:
            peaks.append((t,A[t]))

    (T,P) = zip(*peaks)

    fig.add_subplot(212,sharex=ax1)

    plt.title("Autocorrelation of X")
    plt.ylabel('Autocorrelation Coefficient')
    plt.xlabel('Lag ')
    #plt.xlim([0,M-1])
    #plt.ylim([-1.2,1.2])
    plt.grid()
    plt.plot([0,M-1],[0,0],color='black')
    plt.plot(L,A)
    plt.scatter(T,P,color='red')
    plt.show()
    return peaks


N = 315
X = [math.cos(x/8+20) for x in range(N)]

showAutoCorrelationWithPeaks(X)

print()
```

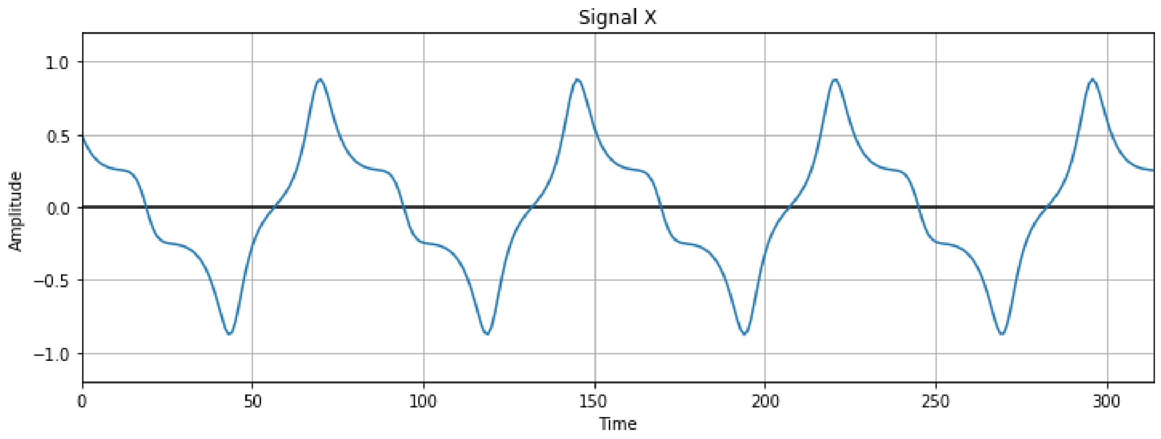


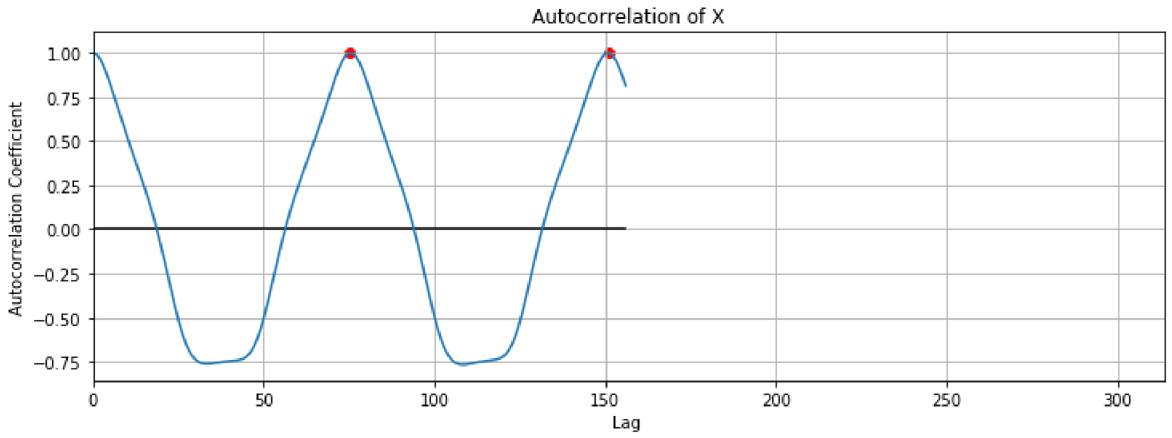


(B) Now a common heuristic is to use the first peak that exceeds a given threshold, say 0.9. For the next example, scan through the peaks returned by the function, and output the first t value for a peak where $A[t] > 0.9$. Express this as "Period found at t time units" for the t you found.

(C) Repeat the same experiment, but using the following signal, and checking for lags from 0 to N/2:

In [21]:
```
N = 315
X = [(math.cos(x/12)/(math.sin(x/4)+2)) for x in range(N)]

peaks = showAutoCorrelationWithPeaks(X)

threshold = 0.9

for (x,y) in peaks:
    if y > threshold:
        print("Period found at", x, "time units.")
        break
```





```
Period found at 75 time units.
```

Problem Nine: Fundamental Frequency of Some Real Signals

In this problem we will try to determine the fundamental frequency of some actual real-world signals. Apply the same technique just shown to determine the fundamental frequency of each of the signals below, and then translate it into Hertz (cycles/second) by the following formula, if the period is found at P time units:

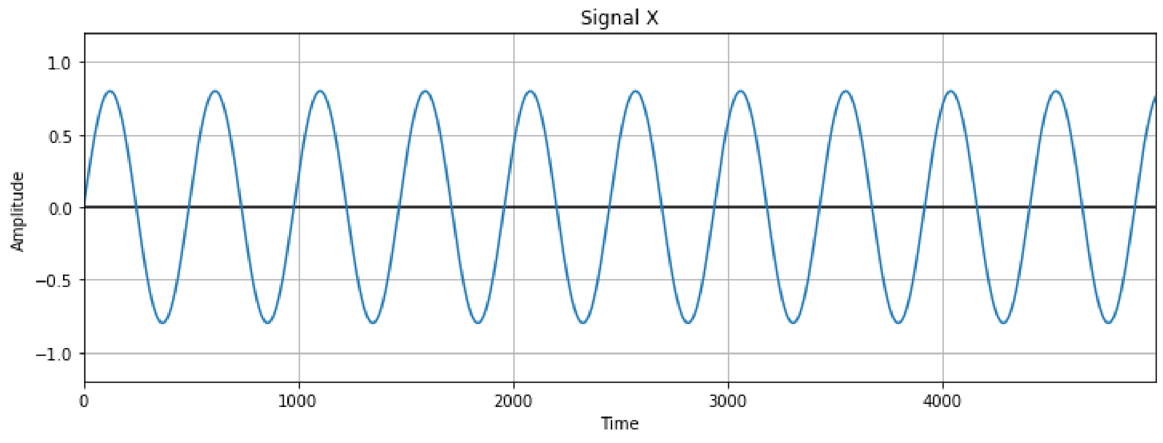
```
    fundamental frequency in Hertz =  (time units / second) / P
```

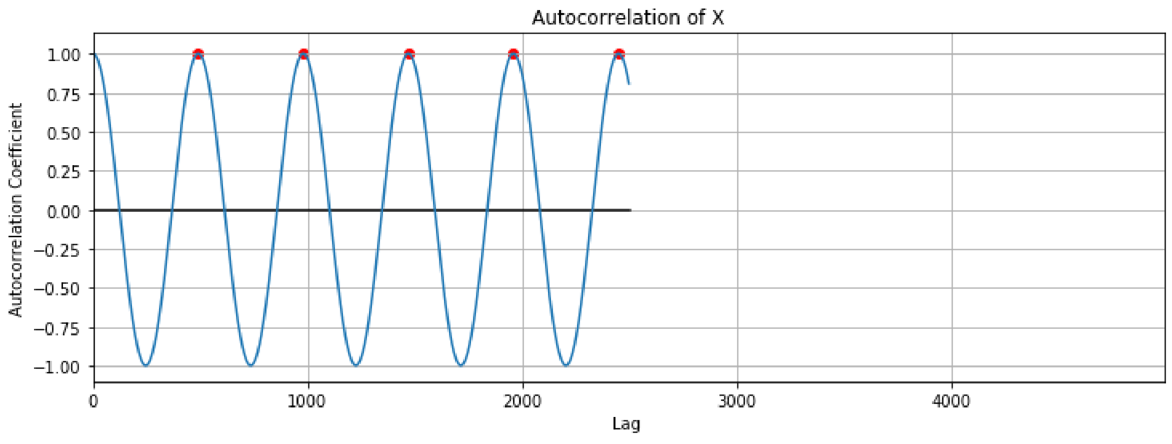
In all the sound files we will experiment with, there are 44100 time units / second, therefore the frequency will be 44100/P Hz.

(A) This signal is an actual sound file, but of a single sine wave. Use a threshold of 0.9 and print out the fundamental frequency. Also, LISTEN to the file using iTunes or other audio player!

In [22]:
```
import array
import contextlib
import wave

def readWaveFile(infile,withParams=False,asNumpy=False):
    with contextlib.closing(wave.open(infile)) as f:
        params = f.getparams()
        frames = f.readframes(params[3])
    return list(array.array('h', frames))
```

In [23]:
```
# (A)

X = readWaveFile("MysterySignal01.wav")[:5000]
X = [(x/32767) for x in X]

peaks = showAutoCorrelationWithPeaks(X)

threshold = 0.9

for (x,y) in peaks:
    if y > threshold:
        print("Period found at", x, "time units.")
        print("Fundamental Frequency = ", round4(44100/x), "Hz")
        break
```

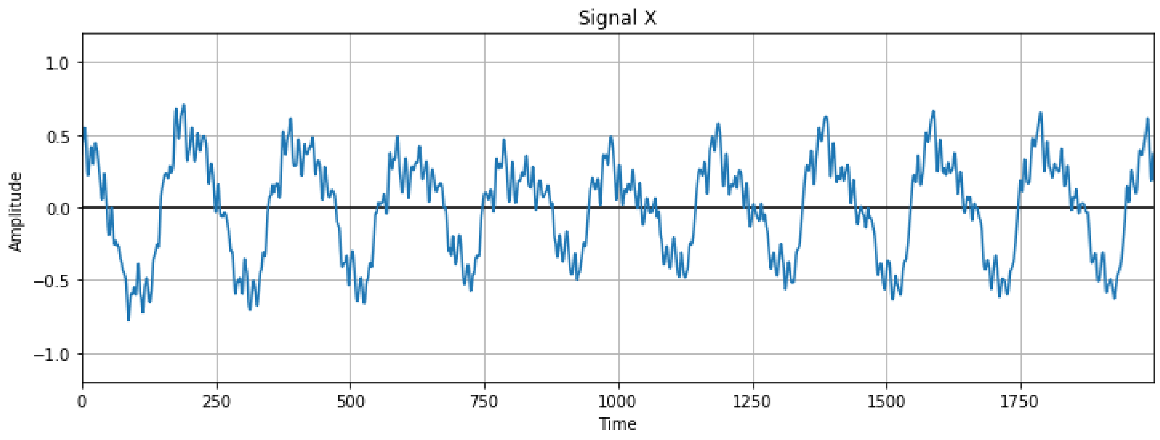


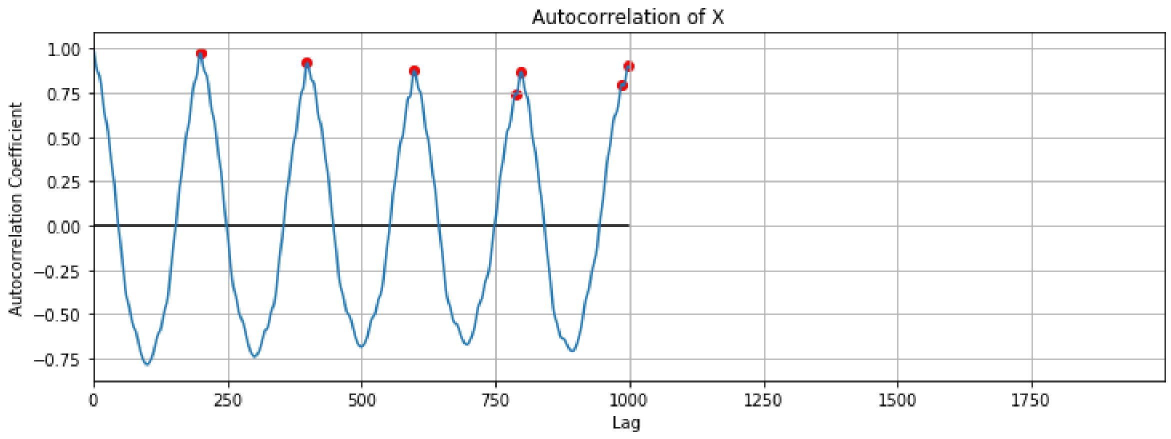


```
Period found at 490 time units.
Fundamental Frequency =  90.0 Hz
```

(B) The next file is of my 1959 Martin steel-string guitar, where I'm playing the low A string. Determine the fundamental frequency of the signal. Use a threshold of 0.9. Listen to the file as well!

In [24]:
```
X = readWaveFile("SteelString.wav")[1000:3000]
X = [(x/32767) for x in X]

peaks = showAutoCorrelationWithPeaks(X)

threshold = 0.9

for (x,y) in peaks:
    if y > threshold:
        print("Period found at", x, "time units.")
        print("Fundamental Frequency = ", round4(44100/x), "Hz")
        break
```

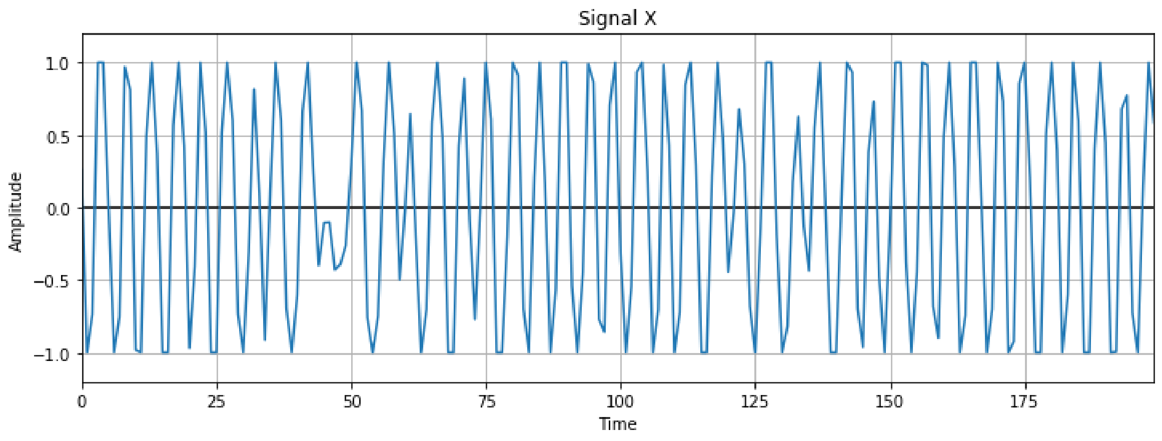


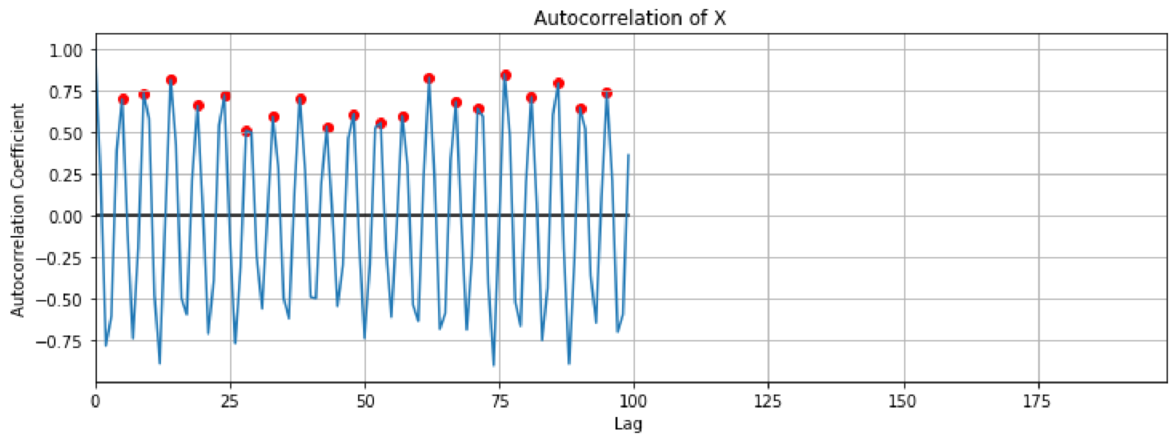


```
Period found at 200 time units.
Fundamental Frequency =  220.5 Hz
```

(C) This is a recording of a bell; unfortunately, we can not always use such a high threshold as 0.9; for this one, you need to set the threshold to around 0.7.

In [25]:
```
X = readWaveFile("Bell.wav")[1000:1200]
X = [(x/32767) for x in X]

peaks = showAutoCorrelationWithPeaks(X)

threshold = 0.7

for (x,y) in peaks:
    if y > threshold:
        print("Period found at", x, "time units.")
        print("Fundamental Frequency = ", round4(44100/x), "Hz")
        break
```



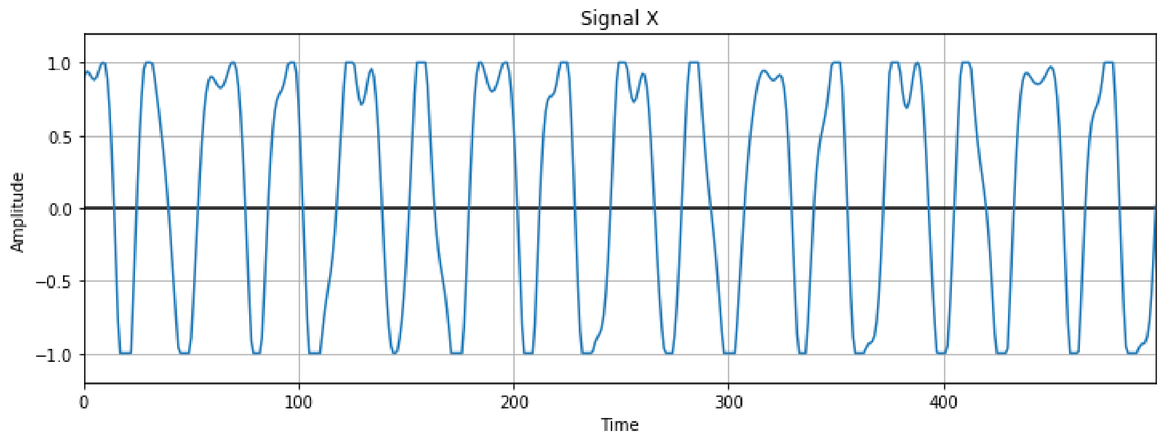


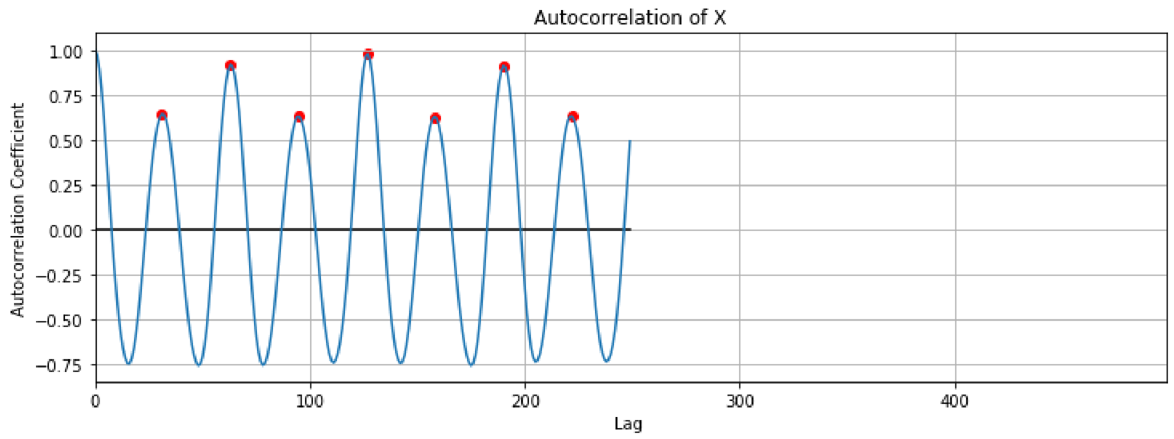
```
Period found at 5 time units.
Fundamental Frequency =  8820.0 Hz
```

(D) This is a recording of the radio signal of the pulsar PSR B1937+21, the dead neutron star left after a supernova, and which has the mass of about our sun but the size of Boston, rotating rapidly and emitting pulses of radiation. How fast is it rotating? Fast enough that the surface of the pulsar is moving at 1/7th the speed of light! And this is only the *second fastest* pulsar that has been observed!

Run the code and find out its rotation speed in Hz. Use a threshold of 0.9.

In [26]:
```
# (D)
X = readWaveFile("pulsar.wav")[:500]
X = [(x/32767) for x in X]

peaks = showAutoCorrelationWithPeaks(X)

threshold = 0.9

for (x,y) in peaks:
    if y > threshold:
        print("Period found at", x, "time units.")
        print("Fundamental Frequency = ", round4(44100/x), "Hz")
        break
```





```
Period found at 63 time units.
Fundamental Frequency =  700.0 Hz
```