Universidad Nacional Autónoma de México
Instituto de Investigación en Matemáticas Aplicadas y Sistemas
Posgrado en Ciencia e Ingeniería de la Computación

Reconocimiento de Patrones

Artemio Baños Gallardo

Tema: Filtro de Wiener y Adaptables

Ejercicio 1.

En este ejercicio se logró generar la señal

x(n) = m1*x(n - 1) + m2*x(n - 2) + v(n) = 0.6530*x(n - 1) - 0.7001*x(n - 2) + v(n)

dónde:

x(n) son las observaciones
m1 y m2 son coeficiente
v(n) es ruido blanco con media cero
x(-1) = x(-2) = 0
n es el momento actual de la observación

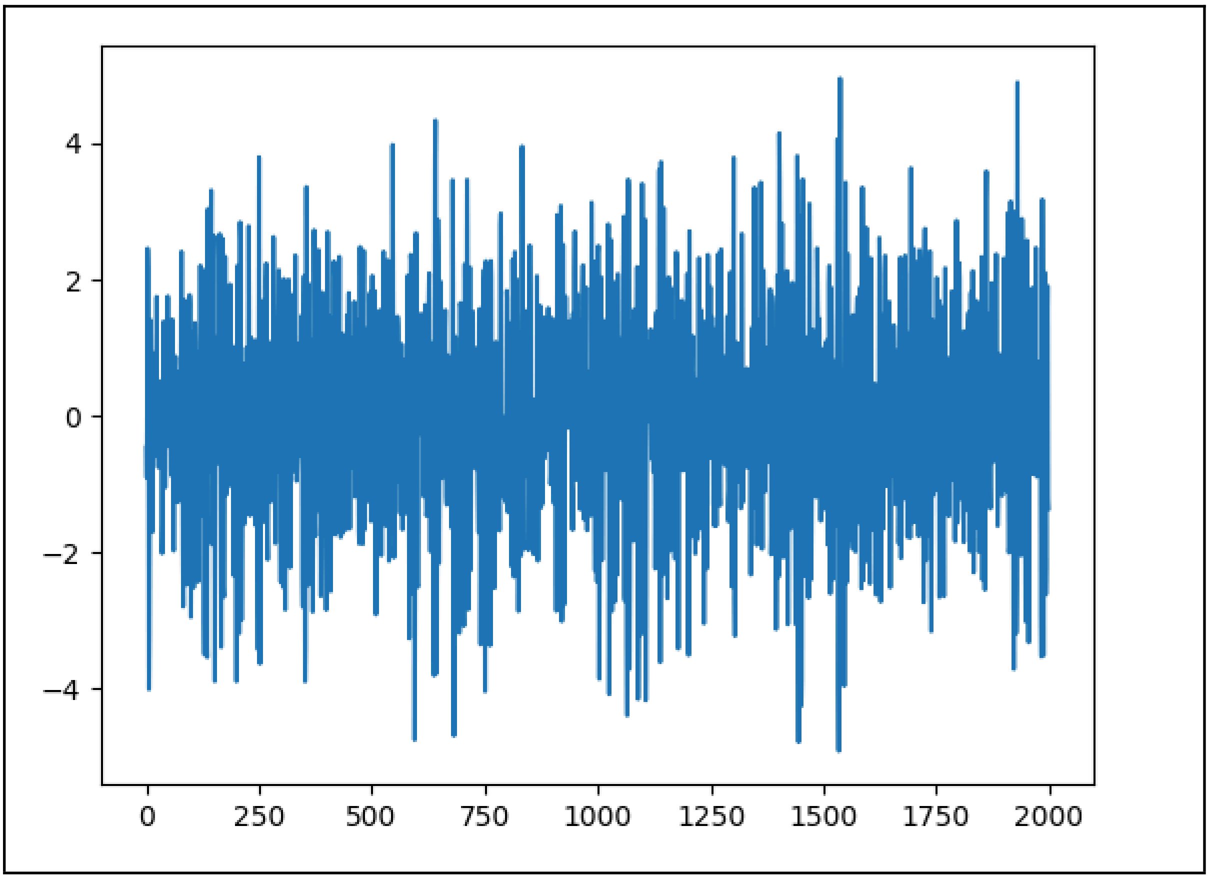

Imagen 1. Señal de observación x(n) con coeficientes {0.6530, -0.7001}

Ejercicio 2.

En este ejercicio se predijeron los coeficientes de la señal de observación x(n) usando el método de filtro de Wiener para predecir coeficientes usando la matriz de correlación y el vector de correlación.

En la imagen 2 se puede observar el poder del predictor de Wiener. Ahí los coeficientes m1 y m2 son muy cercanos a los de la señal de observación.

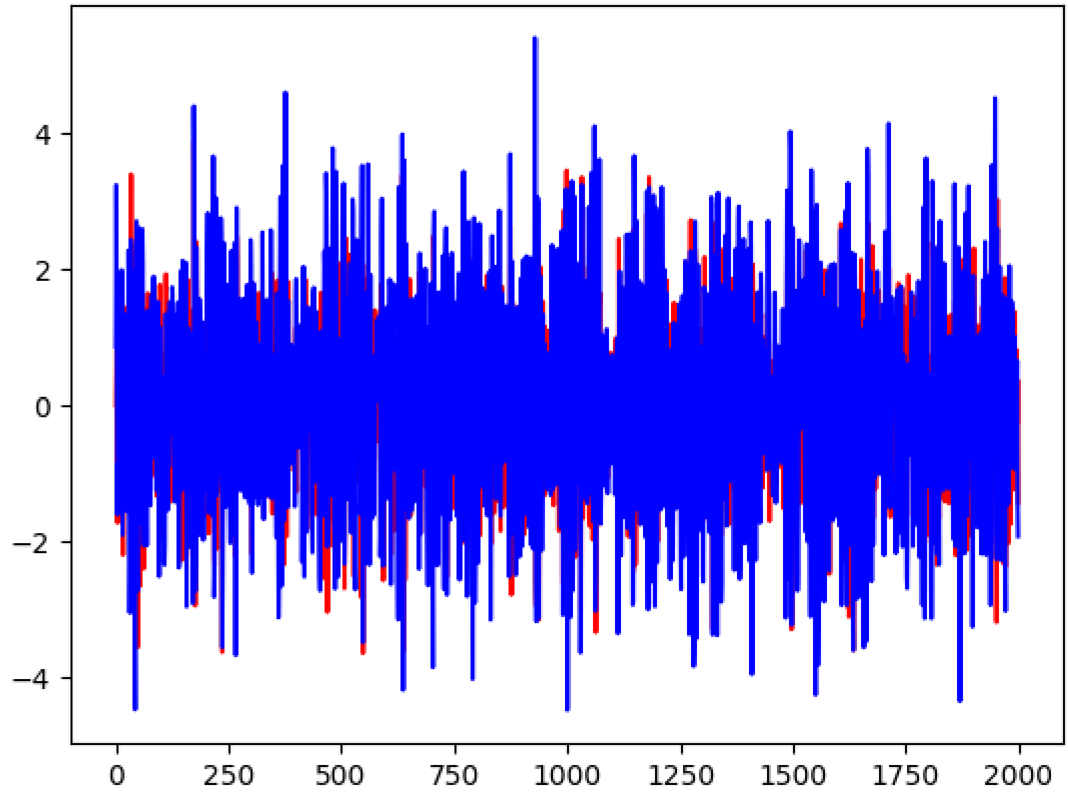

Imagen 2. La señal color rojo es la predecida con los coeficientes de Wiener {0.69183093, -0.71676522}. La señal azul es la señal x(n) {0.6530, -0.7001}

Código

```
# Ejercicio 1
x = getFunction(num_samples, m1, m2, noise)

result = optimalWeightVector(x, numWeinerScalars=[-1, -2],
samples=2000)
print("Wiener Filter Coefficients: {0}".format(result))
```

```
rowsB = len(result)
colsB = len(result[0])

y = [] * 2000
y.insert(0, linearFunction(result[0][0], result[0][1], 0, 0, 0))
y.insert(1, linearFunction(result[0][0], result[0][1], x[0], 0, 0))
for n in range(2000):
  if (n > 1):
    actual = linearFunction(result[0][0], result[0][1], x[n - 1], x[n -
2], 0)
    y.insert(n, actual)

# Plot the functions separately
plt.plot(y, 'r', label='Wiener Predictor')
plt.plot(x, 'b', label='Observation')
plt.show()
```

dónde optimalWeighVector esta dada por:

```
def crossCorrelation(xObservations, filterWeights=[], samples = 8):
  return [singleCrossCorrelation(xObservations, k, samples) for k in
filterWeights]

def singleCrossCorrelation(xObservations, k, samples = 8):
  resultToSum = [multiple(xObservations, i, k) for i in range(samples)]
  return sum(resultToSum)/samples

def multiple(xObservations, n, k):
  return xObservations[n + k]*xObservations[n]

def autoCorrelationMatrix(xObservations, filterWeights = [], samples =
8):
  size = len(filterWeights)
  ac = np.zeros([2, 2], dtype=float)
  # print(range(size))
  for row in range(size):
    for col in range(size):
      k = -row + col
      # print("I[{0}, {1}]: -{0} + {1} = {2}".format(row, col, k))
      sc = singleCrossCorrelation(xObservations, k, samples)
      # print(sc)
      ac[row][col] = sc
  return ac
```

```
def optimalWeightVector(xObservations, numWeinerScalars = [], samples = 8):
  ac = autoCorrelationMatrix(xObservations, numWeinerScalars, samples - len(numWeinerScalars))
  # print("Autocorrellation matrix: " + str(ac))

  aci = np.linalg.inv(ac)
  # print("Autocorrellation matrix inversed: " + str(aci))

  ccm = crossCorrelation(xObservations, numWeinerScalars, samples - len(numWeinerScalars))
  # print("Wiener coefficients T: " + str(ccm))

  # ccmMatrix = np.array(ccm).reshape(1, len(ccm))
  # print("Wiener coefficients: " + str(ccmMatrix))

  ccmMatrix = np.array(ccm).reshape(len(ccm), 1)
  # # print("Wiener coefficients: " + str(ccmMatrix))
  return [multiplyMatrix(aci, ccmMatrix)]
```

y multiplyMatrix por:

```
def multiplyMatrix(A, B):
  rowsA = len(A)
  colsA = len(A[0])

  rowsB = len(B)
  colsB = len(B[0])

  if  colsA != rowsB:
    return "Lenght mismatch A[{0}, {1}] vs B[{2}, {3}]. Cannot multiply A and B.".format(str(rowsA), str(colsA), str(rowsB), str(colsB))

  C = np.zeros((rowsA, colsB), dtype = float)
  for row in range(rowsA):
    for col in range(colsA):
      for elt in range(colsB):
        C[row, elt] += A[row, col] * B[col, elt]

  return C
```

Ejercicio 3.

En este ejercicio intentaré predecir los coeficientes usando un filtro predictivo utilizando la técnica LMS (Least Mean Square). El algoritmo LMS pretende ir reduciendo el error entre la señal de observación x(n) y la señal predecida y(n) en el tiempo

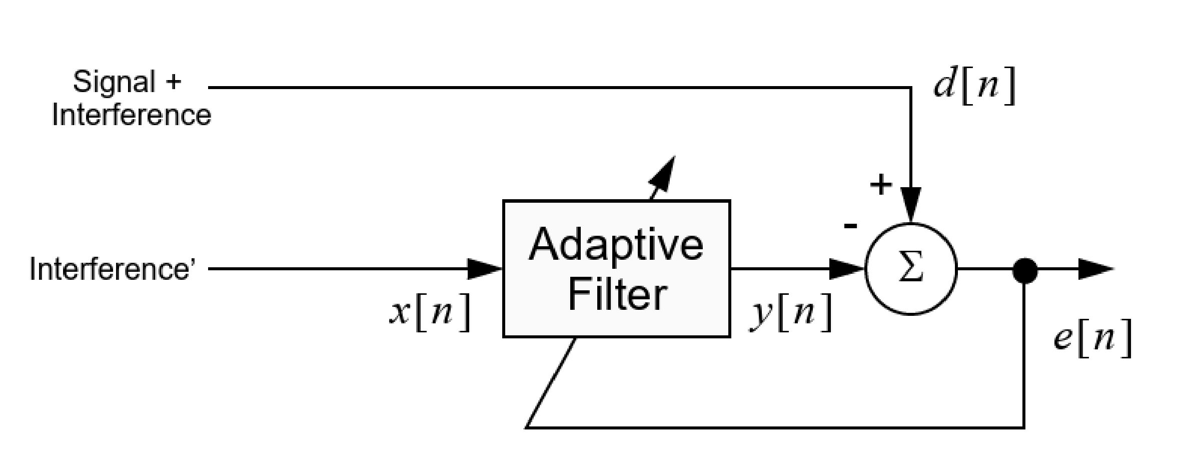

Imagen 3. Arquitectura de filtro LMS adaptativo.

Código

```
import numpy as np
import pre_processing as pp


def lms(n, w, mu, X):
    xn = np.array([pp.generateObservations2(n - 1, X),
                   pp.generateObservations2(n - 2, X)])
    E = error(n, w, X)
    print("W[n] = w[n - 1] - MU*x[n - 1]*E(n - 1) = ")
    W = w + mu*xn*E
    W2 = np.array([0.0, 0.0])
    for k in range(len(w)):
        W2[k] = w[k] + xn[k]*E*mu
    return [W, E]


def error(n, w, X):
    xn = pp.generateObservations2(n, X)
    xn_1 = pp.generateObservations2(n - 1, X)
    xn_2 = pp.generateObservations2(n - 2, X)
    print("error[n] = x[n] - (w[1]*x[n - 1] + w[2]*x[n - 2]) = ")
    estimacion = w[0]*xn_1 + w[1]*xn_2
    print("error[{0}] = {1} - ({2}*{3} + {4}*{5}) = {6}"
```

```
        .format(n + 1, xn, w[0], xn_1, w[1], xn_2, xn - estimacion))
    return xn - estimacion


def adaptiveWeinerFilter(wa, mu, X):
    for n in range(1, len(wa)):
        wa[n, :], errors = lms(n - 1, wa[n - 1, :], mu, X)
    return [wa, errors]


def predictedSignal(x, wa, mu):
    wa, errors = adaptiveWeinerFilter(wa, mu, x)
    yn = np.zeros(pp.num_samples)
    W1 = wa[pp.num_samples - 1, 0]
    print(W1)
    W2 = wa[pp.num_samples - 1, 1]
    print(W2)
    yn[0] = pp.linearFunction([W1, W2], [0, 0], 0)
    yn[1] = W1*x[0] + 0
    for n in range(2, len(wa)):
        yn[n] = W1*x[n - 1] + W2*x[n - 2]
    return yn
```

generateObservations es

```
def linearFunction(m, x, noise):
    return sum(m[k]*x[k] for k in range(len(m))) + noise


def generateObservations(n, noise, m):
    if n == 0:
        return noise[0]

    N = n + 1
    x = np.zeros(N)
    x[0] = noise[0]
    x[1] = linearFunction(m, [x[0], 0], noise[1])

    if n == 1:
        print("x[{0}] = {1}*{2} + {3}*{4} + {5} = {6}".format(n,
```

```
            m[0], x[n - 1], m[1], 0, noise[n], x[n]))
        return x[n]

    for i in range(2, N):
        x[i] = linearFunction(m, [x[i - 1], x[i - 2]], noise[i])
        print("x[{0}] = {1}*{2} + {3}*{4} + {5} = {6}".format(i,
              m[0], x[i - 1], m[1], x[i - 2], noise[i], x[i]))

    return x[n]
```

Resultados

Para Mu = 0.5

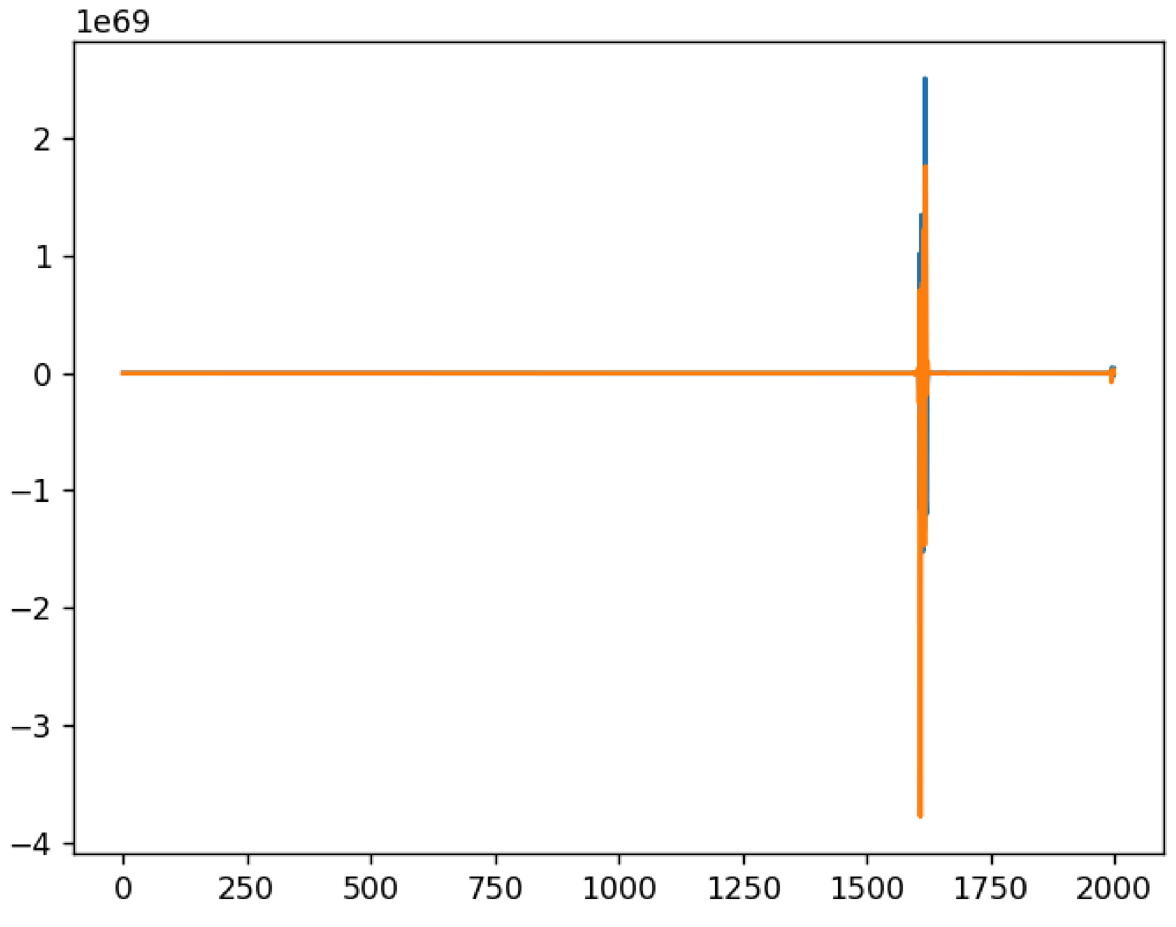

Imagen 4. Los coeficientes predecidos se hacen muy grandes `[4.21507976e+67 3.89444846e+66]`

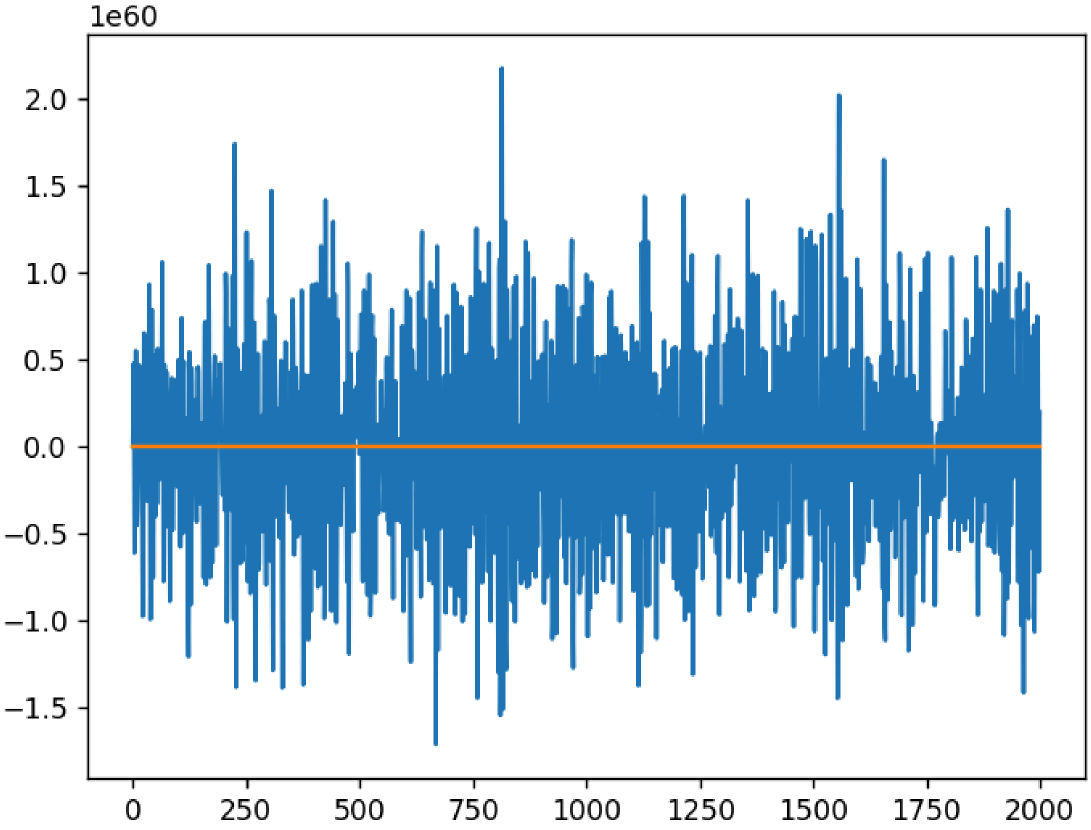

Imagen 5. En naranja es la señal x(n) o señal de observación. En azul es la señal predecida a través de LMS

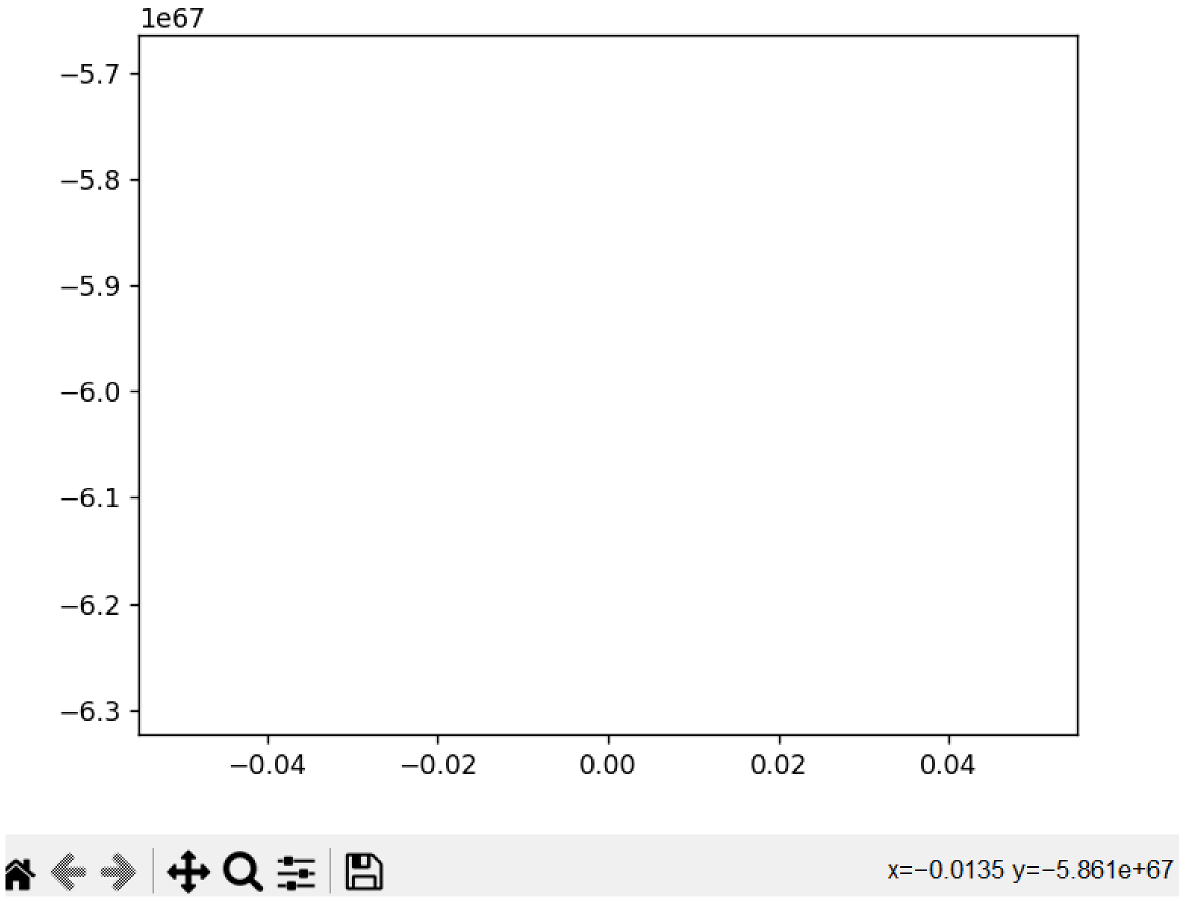

Imagen 6. Los valores del error también son grandes.

Para Mu = 0.1

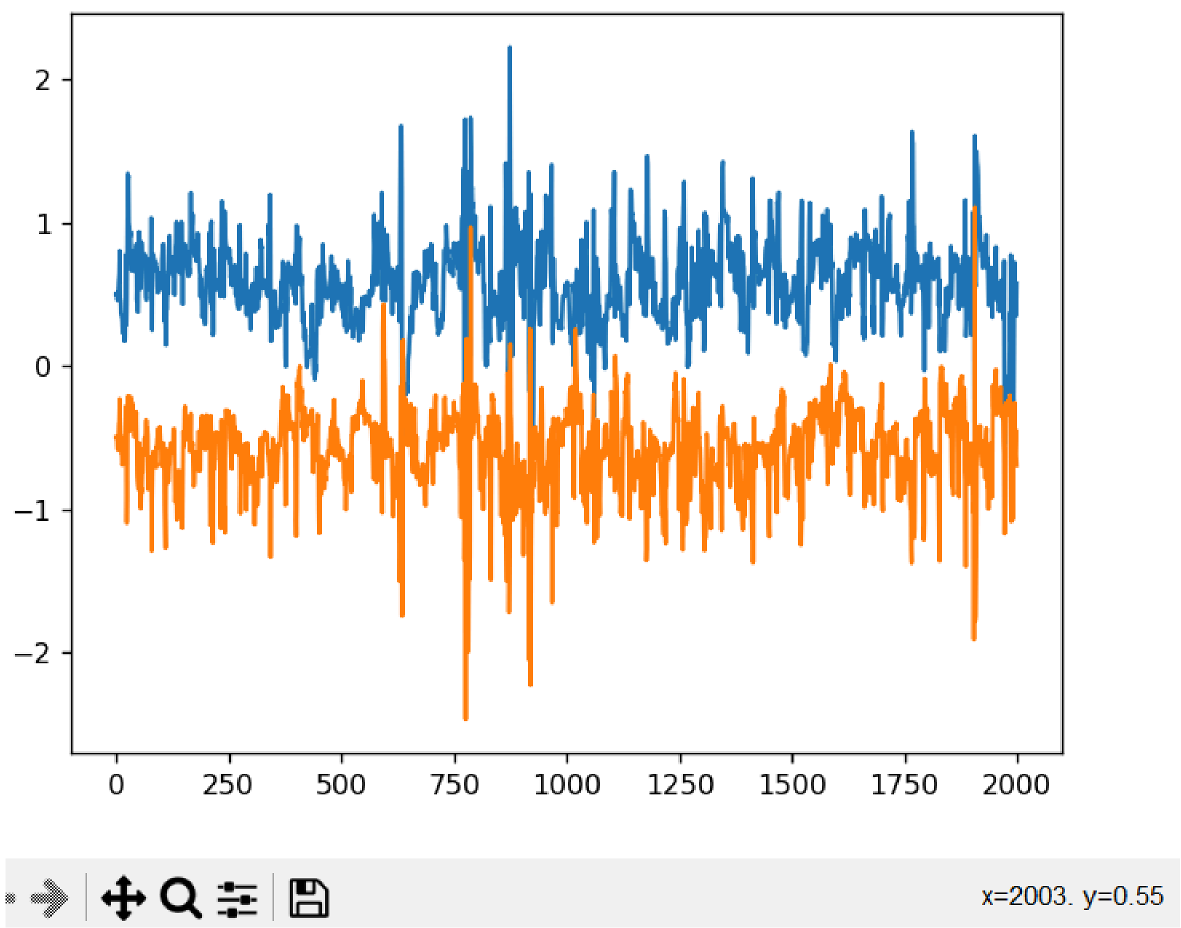

Imagen 7. Para Mu=0.1 el filtro se acerca mucho a la señal de observación con valores [
0.57480159 -0.69737135]

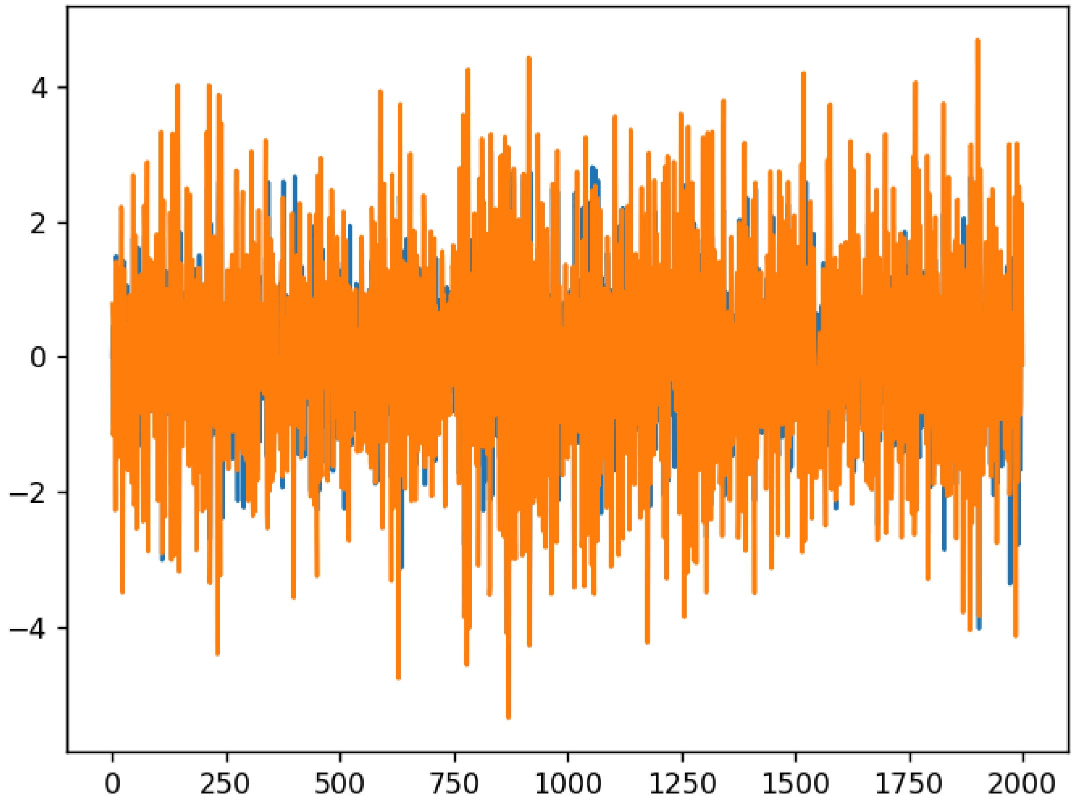

Imagen 8. En naranja se presenta la señal producida usando el predictor LMS. En azul la señal de observación

Referencias:

1. **Dra. María del Pilar Gómez Gil [2017]. Procesamiento digital de señales Semana 11. Filtros Adaptivos LMS. Puede encontrarse en [S10-FiltrosLMS.pdf (inaoep.mx)]**
2. **Universidad Austral de Chile. Estimadores Adaptativos Part I. Puede encontrarse en [[INFO183] Estimadores adaptivos - Filtro LMS (youtube.com)]**