Universidad de Sonora

División de Ciencias Exactas y Naturales

Física computacional

Actividad 7. Reconstruyendo la señal

Cañez Miranda Paul Donaldo
Profesor: Carlos Lizárraga Celaya

28 de Abril del 2017

Resumen

En la práctica anterior aprendimos a descomponer una señal con una transformada discreta de Fourier (DFT) para obtener los modos principales de las mareas. En esta práctica haremos el proceso inverso, a partir de los modos encontrados reconstruiremos la señal çompleta"de la marea y la compararemos con la señal original. Al final calcularemos el error relativo.

Introducción

El análisis de Fourier los utilizamos para obtener las componentes principales de las mareas. Lo que hace es aproximar, mediante la suma de senos y cosenos, las características principales de las ondas en las mareas; amplitud, frecuencia y fase.
Las mareas analizadas son las de los sitios: Isla de Cedros, Baja California y de Nawiliwili, Hawai; descargados del Cicese y Noaa, respectivamente.

Procedimiento

Primero realizaremos los pasos para este sitio. Al igual a prácticas pasadas, lo primero es importar las bibliotecas y paquetes utilizados.

```
import pandas as pd
import numpy as np
import matplotlib as plt
import statsmodels.api as sm
import scipy.stats as stats
from pylab import *
from scipy.io import loadmat
```

Después leer el archivo de los datos y definir el nombre de las columnas. Además de introducir el código

```
from datetime import datetime
```

Para definir una columna que contenga la fecha.

Sabemos que las funciones de senos y cosenos que suma la DFT, son funciones dependientes del tiempo, por lo que es necesario agregar una columna con la variable T, que cuenta las horas transcurridas desde el primer dato hasta el último. La definiremos con el siguiente comando:

```
v = np.arange(0.0, 744.0, 1.0)
df['T'] =  pd.Series(v, index =None)
```

Se debe indicar el número de datos en la muestra. Cada muestra tiene diferente número de datos pero el código es igual para ambos casos.

Isla de Cedros, Baja California

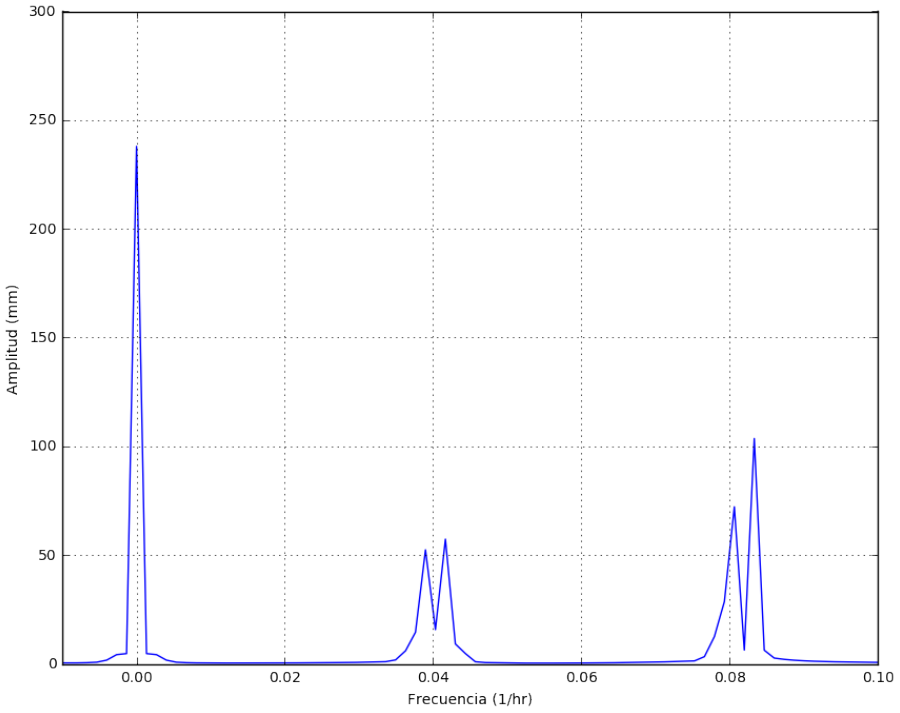
Ahora aplicaremos la DFT con la ayuda de la biblioteca SciPy fftpack de Python para obtener los modos de las mareas en Isla de Cedros:

```
from scipy.fftpack import fft, fftfreq,
fftshift
# number of signal points
N = 744
# sample spacing
T = 1
x = df['date']
y = df['Altura']
yf = fft(y)
xf = fftfreq(N, T)
xf = fftshift(xf)
yplot = fftshift(yf)
import matplotlib.pyplot as plt
plt.plot(xf, 1.0/N * np.abs(yplot))
plt.xlim(-.01, 0.1)
plt.ylim(0,300)
fig=plt.gcf()
fig.set_size_inches(10,8)

plt.xlabel("Frecuencia (1/hr)")
plt.ylabel("Amplitud (mm)")
```

```
plt.grid()
plt.show()
```

Con esto obtenemos la siguiente gráfica donde se muestran las amplitudes y las frecuencias, en los ejes y y x respectivamente:



Es posible obtener el número de dato con su correspondiente amplitud para cada pico. Sólo tomaremos los picos más sobresalientes (de amplítud mayor a 5mm), ya que los demás pueden ser ruido.
Esto lo obtenemos con el siguiente comando:

```
a=np.absolute(yf)/N
#Esto es para encontrar las amplitudes de cada pico, en el dato correspond.
print(np.where(a[:,]>5))
b=a[a[:,]>5]
b
```

Las amplitudes y, las frecuencias y faces correspondientes a dichas amplitudes más notorias fueron:

```
#Amplitud
A0 = np.absolute(yf[int(0),]/N)
A1= 2*np.absolute(yf[27,]/N)
A2= 2*np.absolute(yf[28,]/N)
A3= 2*np.absolute(yf[29,]/N)
A4= 2*np.absolute(yf[30,]/N)
A5= 2*np.absolute(yf[31,]/N)
A6= 2*np.absolute(yf[32,]/N)
A7= 2*np.absolute(yf[58,]/N)
A8= 2*np.absolute(yf[59,]/N)
A9= 2*np.absolute(yf[60,]/N)
A10= 2*np.absolute(yf[61,]/N)
A11= 2*np.absolute(yf[62,]/N)
A12= 2*np.absolute(yf[63,]/N)

#Frecuencia
f1= xf[int(N/2 +27),]
f2= xf[int(N/2 +28),]
f3= xf[int(N/2 +29),]
f4= xf[int(N/2 +30),]
f5= xf[int(N/2 +31),]
f6= xf[int(N/2 +32),]
f7= xf[int(N/2 +58),]
f8= xf[int(N/2 +59),]
f9= xf[int(N/2 +60),]
f10= xf[int(N/2 +61),]
f11= xf[int(N/2 +62),]
f12= xf[int(N/2 +63),]

#Fases
O1= np.angle(yf[27,])
O2= np.angle(yf[28,])
O3= np.angle(yf[29,])
O4= np.angle(yf[30,])
O5= np.angle(yf[31,])
O6= np.angle(yf[32,])
```

```
O7= np.angle(yf[58,])
O8= np.angle(yf[59,])
O9= np.angle(yf[60,])
O10= np.angle(yf[61,])
O11= np.angle(yf[62,])
O12= np.angle(yf[63,])
```

Reconstrucción

Ya tenemos los parámetros que definen la función que vamos a graficar (amplitud, frecuencia, fase). El código para dicha función es:

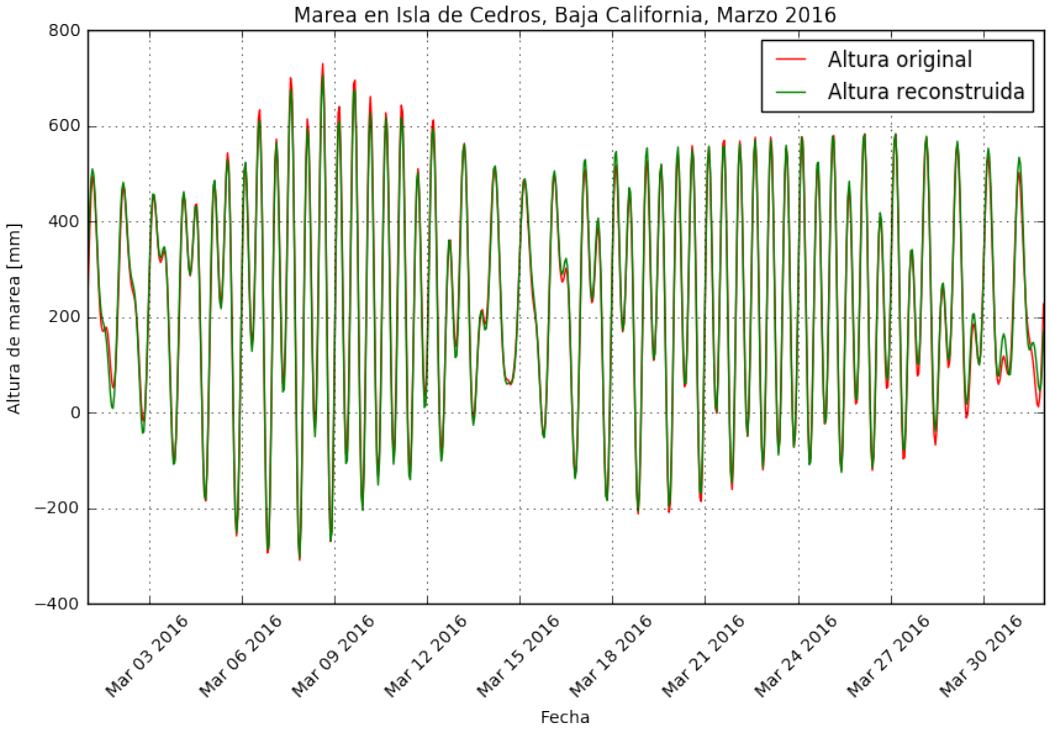
```
y= df['Altura']
w= 2.0*np.pi
a=0
def f(t):  return A0+ (A1*np.cos(w*f1*t+O1) + A2*np.cos(w*f2 *t+O2)
                    + A3*np.cos(w*f3*t+O3) + A4*np.cos(w*f4*t + O4)
                    + A5*np.cos(w*f5*t+O5) + A6*np.cos(w*f6*t + O6)
                    + A7*np.cos(w*f7*t+O7) + A8*np.cos(w*f8*t+ O8)
                    + A9*np.cos(w*f9*t+O9) + A10*np.cos(w*f10*t + O10)
                    + A11*np.cos(w*f11*t+O11) + A12*np.cos(w*f12*t+ O12))
```

Una vez que ya tenemos la función, sólo falta graficar ambas curvas; la señal original que es la que descargamos y la reconstruida. Ambas en una sola gráfica para tener una mejor visión de la aproximación. El siguiente código es para graficar dichas curvas. Nótese que para la curva reconstruida utilizamos el parámetro T.

```
plt.plot(df[u'date'], df[u'Altura'], 'r', label ="Altura original")
plt.plot(df['date'], f(df['T']), 'g', label='Altura reconstruida')
plt.ylabel('Altura de marea [mm]')
plt.xlabel('Fecha')
plt.title('Marea en Isla de Cedros, Baja California, Marzo 2016')
plt.grid(True)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)

# Con esto se ponen las leyendas.
plt.plot(label = "Altura original")
plt.plot(label = "Altura reconstruida")
plt.legend()
```

```
fig = plt.gcf()
fig.set_size_inches(10, 6)
plt.show()
```



El error relativo de nuestra aproximación es:

```
print(sum(abs(y-y1)**2) / sum(abs(y)**2))
```

$$0{,}0025653689159128128 \approx 0{,}26\,\%$$

Nawiliwili, Hawai

El procedimiento para este sitio, es análogo que en el caso de Isla de Cedros. Los únicos cambios son en los datos. Además de que las amplitudes aquí se dan en metros, a diferencia del caso anterior donde estaba en milímetros.

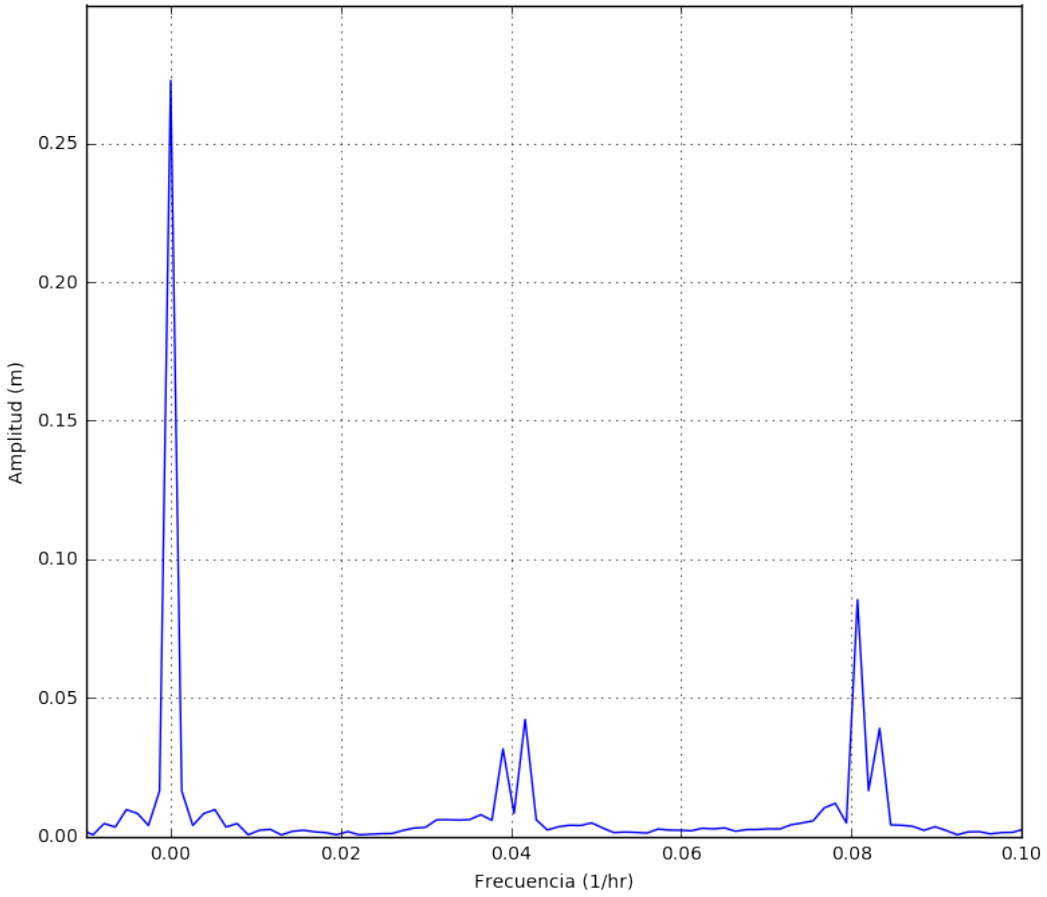
Para la aplicar la DFT:

```
from scipy.fftpack import fft, fftfreq, fftshift
# number of signal points
N = 768
# sample spacing
T = 1
x = df['Fecha']
y = df['Nivel del mar']
yf = fft(y)
xf = fftfreq(N, T)
xf = fftshift(xf)
yplot = fftshift(yf)
import matplotlib.pyplot as plt
plt.plot(xf, 1.0/N * np.abs(yplot))
plt.xlim(-.01, 0.1)
plt.ylim(0,.3)
fig=plt.gcf()
fig.set_size_inches(9,8)

plt.xlabel("Frecuencia (1/hr)")
plt.ylabel("Amplitud (m)")


plt.grid()
plt.show()
```

Obteniendo la siguiente gráfica:



Para obtener las amplitudes más sobresalientes y su respectivo número de dato, introducimos el código:

```
a=np.absolute(yf)/N

#Esto es para encontrar las amplitudes de cada pico, en el dato correspond.
print(np.where(a[:,]>0.01))
b=a[a[:,]>0.01]
b
```

De aquí obtenemos los parámetros que nos interesan para nuestra función:

```
#Amplitud
A0 = np.absolute(yf[int(0),]/N)
A1= 2*np.absolute(yf[1,]/N)
```

```
A2= 2*np.absolute(yf[30,]/N)
A3= 2*np.absolute(yf[32,]/N)
A4= 2*np.absolute(yf[59,]/N)
A5= 2*np.absolute(yf[60,]/N)
A6= 2*np.absolute(yf[62,]/N)
A7= 2*np.absolute(yf[63,]/N)
A8= 2*np.absolute(yf[64,]/N)

#Frecuencia
f1= xf[int(N/2 +1),]
f2= xf[int(N/2 +30),]
f3= xf[int(N/2 +32),]
f4= xf[int(N/2 +59),]
f5= xf[int(N/2 +60),]
f6= xf[int(N/2 +62),]
f7= xf[int(N/2 +63),]
f8= xf[int(N/2 +64),]

#Fases
O1= np.angle(yf[1,])
O2= np.angle(yf[30,])
O3= np.angle(yf[32,])
O4= np.angle(yf[59,])
O5= np.angle(yf[60,])
O6= np.angle(yf[62,])
O7= np.angle(yf[63,])
O8= np.angle(yf[64,])
```

La función nos queda de la siguiente manera:

```
y= df['Nivel del mar']
w= 2.0*np.pi
a=0
def f(t):
    return A0+ (A1*np.cos(w*f1*t+O1) + A2*np.cos(w*f2 *t+O2)
                    + A3*np.cos(w*f3*t+O3) + A4*np.cos(w*f4*t + O4)
                    + A5*np.cos(w*f5*t+O5) + A6*np.cos(w*f6*t + O6)
                    + A7*np.cos(w*f7*t+O7) + A8*np.cos(w*f8*t+ O8))
```

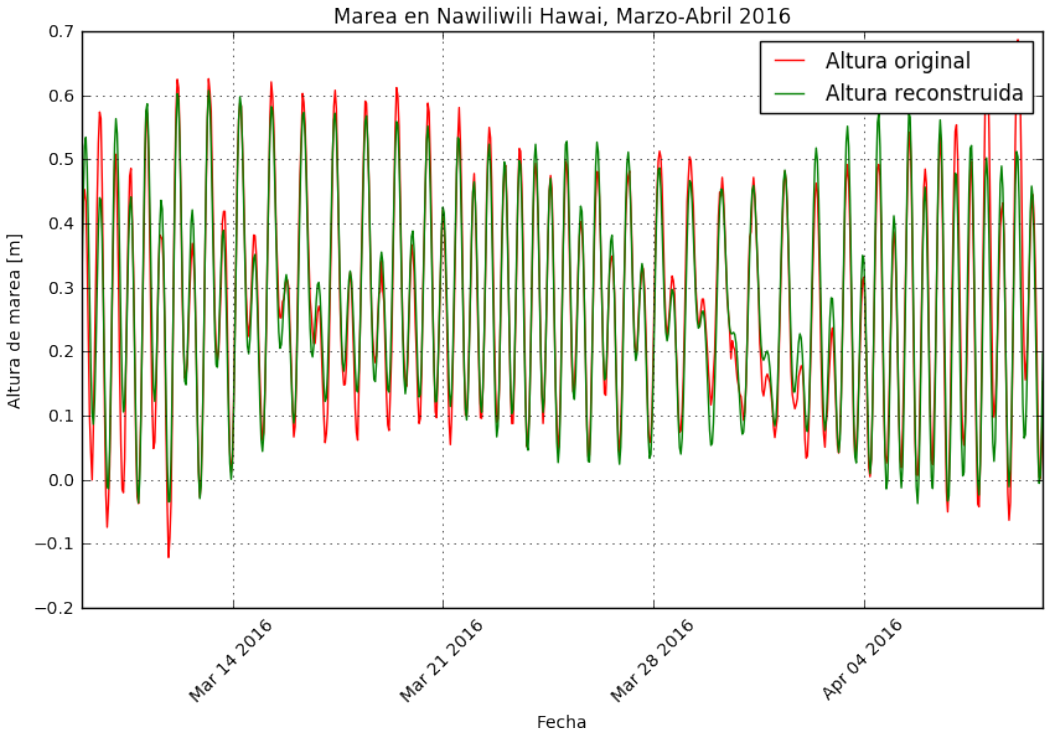
Para la gráfica, el comando es:

```
plt.plot(df[u'Fecha'], df[u'Nivel del mar'], 'r', label ="Altura original")
plt.plot(df['Fecha'], f(df['T']), 'g', label='Altura reconstruida')
plt.ylabel('Altura de marea [m]')
plt.xlabel('Fecha')
plt.title('Marea en Nawiliwili Hawai, Marzo-Abril 2016')
plt.grid(True)
locs, labels = plt.xticks()
plt.setp(labels, rotation=45)

# Con esto se ponen las leyendas.
plt.plot(label = "Altura original")
plt.plot(label = "Altura reconstruida")
plt.legend()

fig = plt.gcf()
fig.set_size_inches(10, 6)
plt.show()
```

Al igual que en el sitio anterior, aquí también se utiliza el parámetro T para graficar la curva reconstruida. La gráfica es:



Con un error relativo de: $0{,}0204629896994288 \approx 2{,}05\,\%$

Conclusión

Las curvas de las señales originales y reconstruidas son muy próximas, lo que indica una buena aproximación. Con esto, podemos ver que con la DFT se obtienen buenos modelos para predecir mareas.
Se podrían obtener mejores resultados si se tomaran en cuenta todo los modos presentes, sin embargo se complicarían mucho los cálculos y la diferencia no sería tan notoria.

Referencias

[1] https://en.wikipedia.org/wiki, Discrete Fourier transform

[2] https://github.com/paulcanez/computacional1/tree/master/Actividad6

[3] http://hyperphysics.phy-astr.gsu.edu/hbasees/Audio/Fourier.html