modelo2

July 5, 2019

In [78]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.fftpack import fft,fftfreq
from scipy.stats import norm, multivariate_normal
import datetime
```

In [79]:
```
sheet_1 = pd.read_excel('Datos ambientales 2016 al 2019 - Piscinas Sur al 5-6-19.xlsx
sheet_2 = pd.read_excel('Datos ambientales 2016 al 2019 - Piscinas Sur al 5-6-19.xlsx
```

In [100]:
```
# (0) Fecha
# (1) Amonio total EFB
# (2) Amoniaco EFB (NH3-N) Mg/L
# (3) Nitrito EFB ( NO2) Mg/L
# (4) Nitrato EFB ( NO3) Mg/L
# (5) pH EFB
# (6) T ř C EFB
# (7) Amonio total SFB
# (8) Amoniaco SFB (NH3-N) Mg/L
# (9) Nitrito SFB ( NO2) Mg/L
# (10) Nitrato SFB ( NO3) Mg/L
# (11) pH SFB
# (12) T ř C SFB
# (13) Alimentacion (Kg)
```

In [172]:
```
# define tiempo como int
t = np.array(sheet_1[sheet_1.columns[0]].astype(int))
# indices de variables distintas al nitrato (piscina 1)
idx = [1,2,4,6]
# define "X" y elimina luego los espacios
X_raw = np.array(sheet_1[sheet_1.columns[idx]])
X = np.where(X_raw==' ', np.nan, X_raw).astype(float)
# define "y" y elimina luego los espacios
y_raw = np.array(sheet_1[sheet_1.columns[3]])
y = np.where(y_raw==' ', np.nan, y_raw).astype(float)
```

In [173]:
```
# mascara para eliminar las filas con datos faltantes
mask_x = set(np.where(np.sum(np.isnan(X), axis=1)==0)[0])
```

```
mask_y = set(np.where(~np.isnan(y))[0])
mask = list(mask_x.intersection(mask_y))
```

In [174]:
```
# ocupa la mascara para dejar solo las filas con datos completos
t_ = t[mask]
y_ = y[mask]
# X considera solo nitrito y pH por ahora
X_ = X[mask, 2:4]
```

In [175]:
```
# likelihood de coeficientes en modelo ARMA, recibe:
# - una serie de coeficientes c
# - x, y, t
# - sgm: representa
# modelo arma usado: am=amonio, p=pH, y=nitrito
# y(t) = c0*y(t-2)+c1*y(t-1)+c2*am(t-2)+c3*am(t-1)+c4*am(t)+c5*p(t-2)+c6*p(t-1)+c7*p
def arma_lik(c_coefs, x, y, t, sgm):
    log_lik = 0
    for i in range(2, len(y)-1):
        vect = np.concatenate((y[i-2:i].reshape(1, y[i-2:i].size), x[i-2:i+1, :].resh
        y_pred = (np.dot(c_coefs, vect.T) * (t[i] - t[i-1]) / t[i])

        log_lik += np.log(norm(y[i], sgm).pdf(y_pred))# / (len(x)*x[i-2:i, :].size)
    return log_lik
```

In [176]:
```
# para partir, se usa c0, c1 correspondientes a la autocorrelacion de y
autocorr = [0.75, 0.6]
# para partir, se usa c2 a c7 correspondientes a la correlaciones de las variables c
temp_corr = [0.32, 0.32, 0.32, -0.39, -0.39, -0.39]
# lista que guarda los coeficientes
c_coefs_list = [np.concatenate((autocorr, temp_corr))]
c_var_matrix = np.diag([2]*8)
```

In [177]:
```
log_lik0 = arma_lik(c_coefs_list[0], X_, y_, t, sgm=0.2)
```

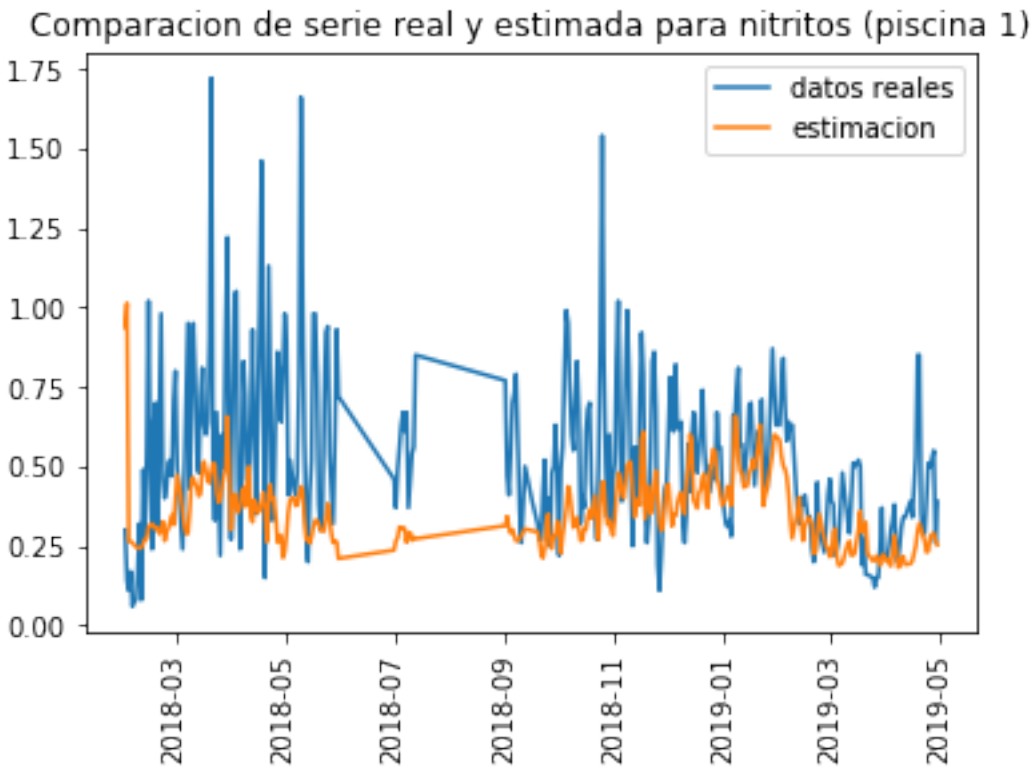
In [178]:
```
N = 100

for _ in range(N):
    # multivar sirve para obtener la el siguiente candidato de c_coefs
    multivar = multivariate_normal(c_coefs_list[-1], c_var_matrix)
    c_coefs_ast = multivar.rvs()
    # se calculan las likelihoods con el ultimo set de coeficientes y el nuevo candi
    p_ast = arma_lik(c_coefs_ast, X_, y_, t, 0.2)
    p_t = arma_lik(c_coefs_list[-1], X_, y_, t, 0.2)
    # se calculan las q del set de coeficientes antiguo y del nuevo candidato
    q_ast_t = multivar.pdf(c_coefs_ast)
    #multivar = multivariate_normal(c_coefs_ast, c_var_matrix)
    q_t_ast = multivar.pdf(c_coefs_list[-1])
    # se calcula a para comparar con u
    a = min(p_ast + np.log(q_t_ast) - (p_t + np.log(q_ast_t)), 1)
```

```
            u = np.random.uniform(0, 1)
            if u < np.exp(a):
                c_coefs_list.append(c_coefs_ast)
```

In [179]:
```
# calcula la serie resultante dados:
# - los coeficientes c
# - la matriz X
# - el tiempo que corresponde a cada una de las mediciones
# - dos datos iniciales para y
def series(X, c_coefs, t, y0, y1):
    # define la lista y donde se guardan los valores de nitrito
    y = [y0, y1]
    # itera calculando el siguiente valor de y segun el modelo ARMA
    for i in range(2, X.shape[0]):
        # obtiene el vector de valores
        vect = np.concatenate((np.array(y[i-2:i]).reshape(1,2), X[i-2:i+1, :].reshape
        # multiplica por el vector de coeficientes, considerando el tiempo entre mue
        y_pred = (np.dot(c_coefs, vect.T) * (t[i] - t[i-1]) / t[i])[0]
        y.append(y_pred)
    return y
```

In [180]:
```
burn_in = 12
window_size = 4
c_coefs = np.array(c_coefs_list[burn_in::window_size]).mean(axis=0)
serie_1 = series(X_, c_coefs, t, y0=0.25, y1=0.27)
```

In [181]:
```
date = [datetime.datetime.fromtimestamp(d / 1e9) for d in t_]
# plt.figure(figsize=(18, 10))
plt.plot(date, y_, label='datos reales')
plt.plot(date, np.array(serie_1) * 3.75, label='estimacion')
plt.xticks(rotation='vertical')
plt.legend()
plt.title('Comparacion de serie real y estimada para nitritos (piscina 1)');
```



In [171]:
```
log_lik_fin = arma_lik(c_coefs, X_, y_, t, sgm=0.2)
```

In [ ]: