# Herrera_Vial_Proy_2

July 5, 2019

```
In [79]: 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 [80]: 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 [81]: # (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 [82]: # define tiempo como int (cada entero, un dia)
         t = np.array(sheet_1[sheet_1.columns[0]]).astype('timedelta64') / (1000000000*60*60*2
         # indices de variables distintas al nitrato (piscina 1)
         idx = [1, 2, 4, 5, 6, 13]
         # define "X" y elimina luego los espacios
         X_raw = np.array(sheet_1[sheet_1.columns[idx]])
         X_cleaned = 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 [ ]:
```

In [83]:
```
# X considera solo nitrito y pH por ahora
# 0 al 5: amonio, amoniaco, nitrato, pH, T, alimento
idx_params = np.array([2, 3, 5])
X = X_cleaned[:, idx_params]
# mascara para definir las filas con datos suficientes (considerando los idx_params)
mask_x = set(np.where(np.sum(np.isnan(X), axis=1)==0)[0])
mask_y = set(np.where(~np.isnan(y))[0])
mask = np.array(list(mask_x.intersection(mask_y)))
```

In [84]:
```
# likelihood de coeficientes en modelo ARMA, recibe:
# - una serie de coeficientes c
# - x, y, t
# - sgm: representa que tan ajustado debe estar el valor de prediccion con respecto a
# modelo arma usado:
# coefs: matriz de dimension [k_per, n_vars + 1]
#        primera col corresponde a coeficientes a
#        demas columnas corresponden a coeficientes c
#        la ultima fila es la mas cercana al presente
# donde n_vars es el numero de variables que considera x
# ej. n_vars = 2, k = 2
# am = amonio, p = pH, y = nitrito
def arma_lik(coefs, x, y, mask, initial_y, k_per, sgm=0.8):
    log_lik = 0
    h, w = x.shape
    yX = np.zeros((h, w + 1))
    yX[:k_per, 0] = initial_y
    yX[:, 1:] = np.nan_to_num(x)
    for i in range(k_per, h):
        # calcula solo las filas en que hay datos suficientes
        if i in mask and i - 1 in mask:
            yX[i, 0] = np.sum(yX[(i - k_per):i, :] * coefs) / 10
            log_lik += np.log(norm(y[i], sgm).pdf(yX[i, 0])) / 1e20
            #print(np.log(norm(y[i], sgm).pdf(yX[i, 0])) / 1e20)
            #print(y[i], yX[i,0])
    return log_lik
```

In [85]:
```
# tamano matriz en tiempo
k_per = 2
# datos iniciales de y
initial_y = [0.5, 0.5]


# # crea una matriz para los coeficientes a partir de correlaciones y autocorrelacion
# # correlaciones con nitrito
# # nitrito, amonio, amoniaco, nitrato, pH, T, alimento
# corr = np.array([0.75, 0.45, 0.15, 0.5, -0.39, -0.06, -0.07])
# # autocorrelacion nitrito
# autocorr = 0.75
```

```
        #
        # autocorr_vect = np.array([autocorr**(k_per - i - 1) for i in range(k_per)])
        # # se usa esta matriz para partir
        # coef_matrix = np.outer(autocorr_vect, corr)
        # # agrega el indice 0 para considerar los coeficientes "a" (para la salida "y")
        # idx_coefs = np.concatenate(([0], idx_params))
        # define matrix de coeficientes (divide en 100 para probar partir desde efecto bajo)
        #coef_matrix = coef_matrix[:, idx_coefs] / 100


        coef_matrix = np.zeros((k_per, len(idx_params) + 1))
        # obtiene dimension de matriz para luego usarla en reshape
        c_matrix_shape = coef_matrix.shape
        # crea la matrix de covarianzas entre coeficientes
        coef_variance_matrix = np.diag([0.07] * coef_matrix.size)
```

```
In [8]: log_lik0 = arma_lik(coef_matrix, X, y, mask, initial_y, k_per, sgm=1.5)
```

```
In [47]: def metropolis_hastings(X, y, initial_y, mask, coef_variance_matrix, c_matrix_shape, s
             # lista que guarda las posibles matrices de coeficientes que generar el metodo
             coefs_list = [np.zeros(c_matrix_shape)]
             # numero de iteraciones
             # cuenta cantidad de aceptados por el metodo
             j = 0
             for i in range(N):
                 # multivar sirve para obtener el siguiente candidato de c_coefs
                 multivar = multivariate_normal(coefs_list[-1].flatten(), coef_variance_matrix
                 coefs_ast = multivar.rvs()
                 # se calculan las likelihoods con el ultimo set de coeficientes y el nuevo ca
                 p_ast = arma_lik(coefs=coefs_ast.reshape(c_matrix_shape), x=X, y=y,
                                  initial_y=initial_y, mask=mask, k_per=k_per, sgm=sgm)
                 p_t = arma_lik(coefs=coefs_list[-1].reshape(c_matrix_shape), x=X, y=y,
                                initial_y=initial_y, mask=mask, k_per=k_per, sgm=sgm)
                 a = min(np.exp(p_ast - p_t), 1)
                 #print(p_ast, p_t)
                 u = np.random.uniform(0, 1)
                 #print(a, u)
                 if u < np.exp(a):
                     j += 1
                     coefs_list.append(coefs_ast.reshape(c_matrix_shape))
             print(j / N)
             return coefs_list
```

```
In [86]: coefs_list=metropolis_hastings(X, y, initial_y, mask, coef_variance_matrix, c_matrix_s
```

```
1.0
```

```
In [55]: # calcula la serie resultante dados:
         # - los coeficientes c
```

```
    # - la matriz X
    # - el tiempo que corresponde a cada una de las mediciones
    # - datos iniciales para y (largo k_per)
    def make_series(X, coefs, initial_y, k_per):
        h, w = X.shape
        yX = np.zeros((h, w + 1))
        yX[:, 1:] = np.nan_to_num(X)
        yX[:k_per, 0] = initial_y
        for i in range(k_per, h):
            yX[i, 0] = np.sum(yX[(i - k_per):i, :] * coefs) / 10
            #print(yX[i,0])
            #print(yX[(i - k_per):i, :] * coefs)
        return yX[:, 0]
```

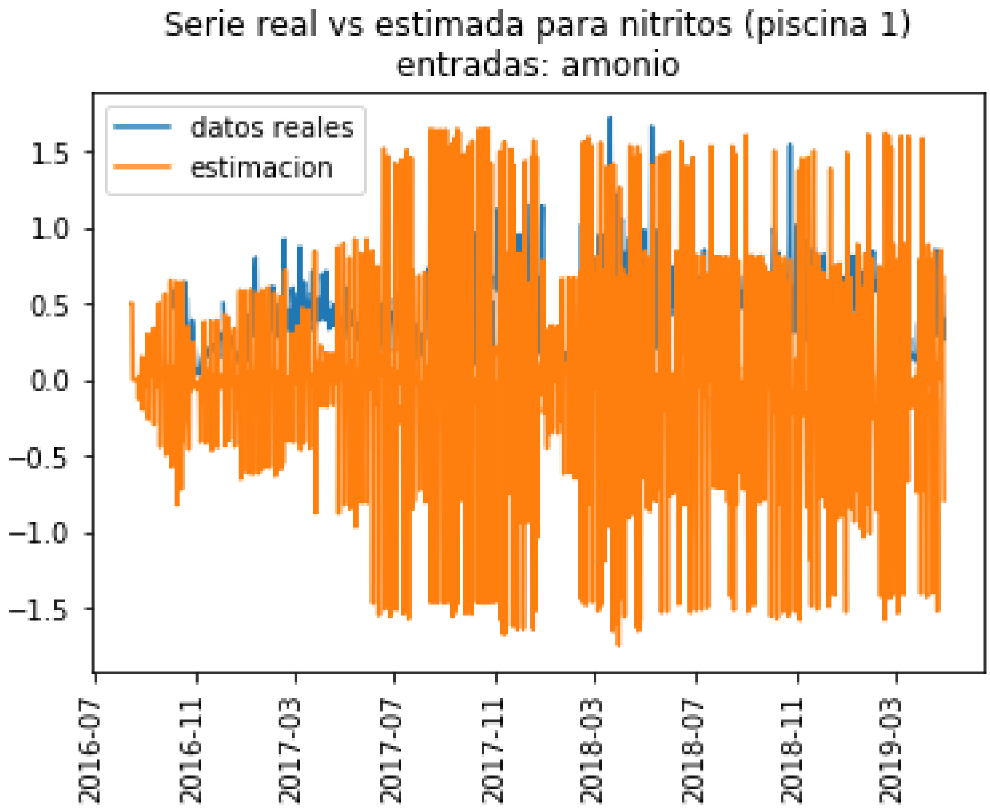
```
In [20]: def get_coefs(coefs_list, burn_in, window_size):
             return np.array(coefs_list[burn_in::window_size]).mean(axis=0)

         burn_in = 2
         window_size = 2
         coefs = get_coefs(coefs_list, burn_in, window_size)

         series = make_series(X, coefs, initial_y=[0.5, 0.5], k_per=k_per)
         coefs
         #print(serie_1)
```

```
Out[20]: array([[-0.5188172 , -0.97600717, -0.62979892,  0.74532163],
                [-0.30844612, -0.10870035,  0.33465305, -0.65647492]])
```

```
In [23]: #print(len(date), t.shape, y.shape, serie_1.shape)
         def plot_series(y, series, t, params=''):
             date = [datetime.datetime.fromtimestamp(d * 60 * 60 * 24) for d in t]
             plt.plot(date, y, label='datos reales')
             plt.plot(date, series, label='estimacion')
             plt.xticks(rotation='vertical')
             plt.legend()
             plt.title('Serie real vs estimada para nitritos (piscina 1)\n' + params);
```

```
In [24]: plot_series(y, series, t, params='entradas: amonio')
         log_lik_fin = arma_lik(coefs, X, y, mask, initial_y, k_per, sgm=0.2)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launche
```




```
In [25]: print(log_lik0, log_lik_fin)
```

```
-9.423319149816175e-18 -inf
```

```
In [26]: ###########################################################################
```

```
In [ ]:
```

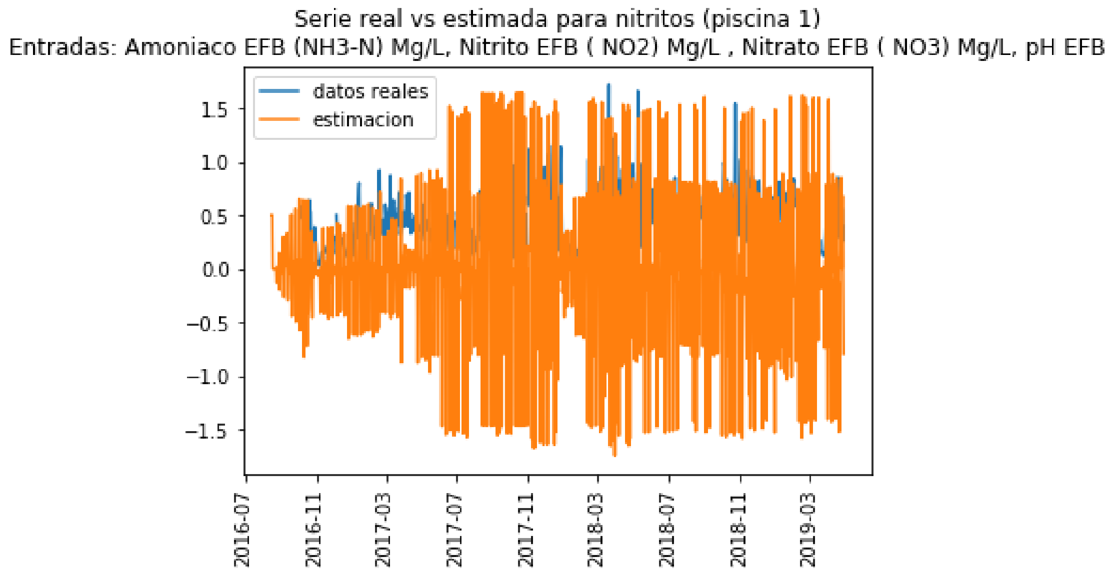
```
In [49]: def main(t, complete_X, idx_params, y, initial_y, k_per, sgm_arma, N, coef_variance,
              burn_in, window_size, plot=False):
             X = complete_X[:, idx_params]
             # mascara para definir las filas con datos suficientes (considerando los idx_param
             mask_x = set(np.where(np.sum(np.isnan(X), axis=1)==0)[0])
             mask_y = set(np.where(~np.isnan(y))[0])
             mask = np.array(list(mask_x.intersection(mask_y)))
             coef_matrix = np.zeros((k_per, len(idx_params) + 1))
             c_matrix_shape = coef_matrix.shape
             log_lik0 = arma_lik(coef_matrix, X, y, mask, initial_y, k_per, sgm_arma)
             coef_variance_matrix = np.diag(np.ones(coef_matrix.size) * coef_variance)
             coefs_list = metropolis_hastings(X, y, initial_y, mask, coef_variance_matrix, c_ma
             #burn_in = min(burn_in, len(coefs_list) / 10)
```

```
        #window_size = min(window_size, (len(coefs_list) - burn_in) / 10)
        coefs = get_coefs(coefs_list, burn_in, window_size)
        log_lik_fin = arma_lik(coefs, X, y, mask, initial_y, k_per, sgm_arma)
        print("Log likelihood: inicial={}, final={}".format(log_lik0, log_lik_fin))
        if plot:
            aux = np.array(['Fecha', 'Amonio total EFB', 'Amoniaco EFB (NH3-N) Mg/L',
                   'Nitrito EFB ( NO2) Mg/L ', 'Nitrato EFB ( NO3) Mg/L', 'pH EFB',
                   'T ř C EFB', 'Amonio total SFB', 'Amoniaco SFB (NH3-N) Mg/L',
                   'Nitrito SFB ( NO2) Mg/L ', 'Nitrato SFB ( NO3) Mg/L', 'pH SFB',
                   'T ř C SFB', 'Alimentacion (Kg)'])[idx_params]
            params = 'Entradas: ' + (', ').join(aux)
            plot_series(y, series, t, params)
        return make_series(X, coefs, initial_y, k_per)
```

```
In [70]: idx_params = np.array([2, 3, 4, 5])
         k_per = 5
         initial_y = [0.5, 0.5, 0.45, 0.45, 0.4]
         coef_variance = 0.07
         sgm_arma = 1.5
         N = 100
         burn_in = 15
         window_size = 12
         coef_variance = 0.07
         # 0 al 5: amonio, amoniaco, nitrato, pH, T, alimento
```

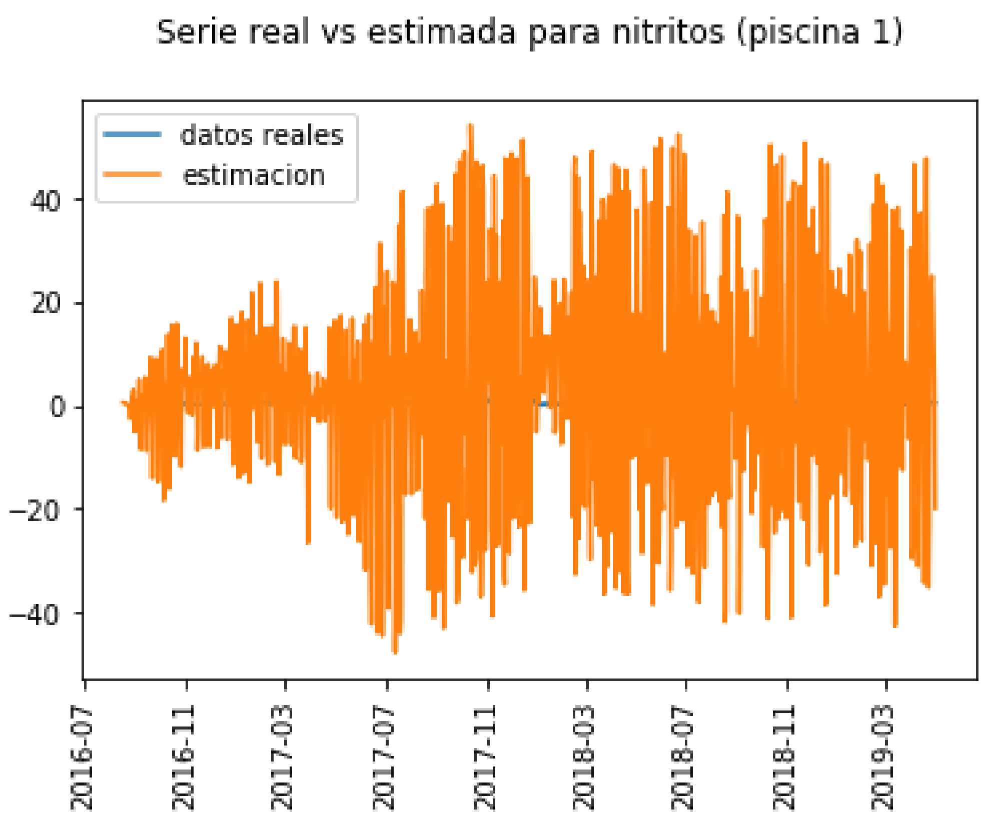
```
In [71]: series1 = main(t, X_cleaned, idx_params, y, initial_y, k_per, sgm_arma, N, coef_variar
                        burn_in, window_size, plot=True)
         #series2 = main(t, X_cleaned, np.array([3, 4]), y, initial_y[:2], k_per=2, sgm_arma, 
         #               burn_in, window_size, plot=True)
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launche
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/ipykernel_launche
  app.launch_new_instance()


0.25
Log likelihood: inicial=-6.386319937719867e-18, final=-6.200116384963182e-16
```




```
In [57]: #lista_series1 = [series1, series2, series3, series4, series5, series6, series7, seri
```

```
In [73]: plot_series(y, series2, t)
```



```
In [ ]:
```