PCA Y ANALOGÍA – GEOMETRÍA COMPUTACIONAL – 2023

PABLO FERNÁNDEZ DEL AMO

1. INTRODUCCIÓN:

El objetivo de esta práctica es analizar los cambios en la temperatura y la altura geopotencial de la atmósfera utilizando archivos NetCDF de los años 2021 y 2022. El análisis se centrará en la identificación de las componentes principales y en el estudio de los días más análogos basados en la distancia euclídea.

2. MATERIAL USADO:

Se utilizaron cuatro archivos NetCDF, dos de temperatura (air*) y dos de altura geopotencial (hgt*) correspondientes a los años 2021 y 2022 proporcionados por un re-análisis climatológico NCEP. Los datos se organizan en 144 longitudes (x), 73 latitudes (y) y 17 niveles de presión (p). El análisis se realizó considerando la atmósfera como un sistema de 6 variables de estado, discretizando las coordenadas x, y y p en 144, 73 y 17 valores posibles respectivamente, y la variable tiempo t en lapsos de 1 día. Para el PCA concretamente para la analogía se utilizó la distancia euclídea con los pesos marcados en el enunciado; todo esto en Python con sus librerías correspondientes.
El archivo utilizado como plantilla ha sido : *GCOM2023-practica_PCA_ANN_plantilla.py*.
Cabe destacar que las funciones downsample_time agrupa las horas que pertenecen a un mismo día en un solo registro (el primero) y daily_average devuelve los archivos de altura geopotencial y temperatura agrupados por fechas, es decir, haciendo la media de los valores que tenemos por cada uno de los cuatro registros que tenemos para cada día (cada uno con una diferencia de 6h)

3. RESULTADOS:

i) Considera el sistema $S = \{a_d, X_d\}^{365}_{d=1}$ formado por los elementos 'días' a_d del año 2021 y las variables de estado $X_d := \{Z_{i,j,k}\}^{i=144,j=73,k=17}_{i=1,j=1,k=1}$ y estima las 4 componentes principales fijando p_k = 500hPa. Represéntalas espacialmente. ¿Qué porcentaje de varianza se explica?

Sea X la matriz que almacena los datos de altura geopotencial de 2021 cuyo nivel de presión es 500hPa. Aplicando el PCA (Analysis of Principal Components) a X y a X^T, encontramos que La varianza explicada de las 4 primeras componentes para X suma hasta casi un 60% de la varianza, siendo la primera componente con un 47,249% la que más explica, seguida de la segunda, tercera y cuarta con un 6,073%, 3,593% y 2,815% , respectivamente.

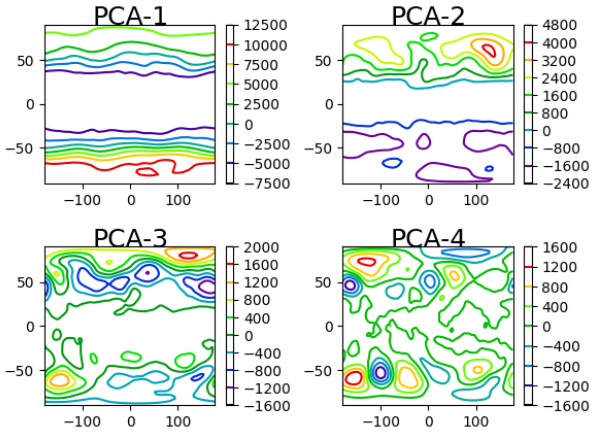

Ilustración 1

En cambio, $Y = X^T$, explica la varianza en más de un **94%, divida en un 88,773% para la primera componente, y un 5,178%, 0,544% y un 0,358% para las tres últimas, respectivamente.**

En la ilustración 1 se observa de manera gráfica las 4 componentes principales resultantes de realizar el PCA sobre los datos previamente menciona. El comportamiento regular que se observa de la altura geopotencial a través de la primera componente es reflejo del alto porcentaje de explicación que tiene esta componente de la varianza. Por lo que se puede determinar a través de la latitud. En la segunda componente principal, al igual que en la primera, se encuentra una zona alrededor del 0 con Z constante, a diferencia de la tercera y cuarta componente donde el comportamiento es bastante más irregular.

ii) Considera un subsistema $\sigma \subset S$ delimitado* por $x \in (-20^\circ, 20^\circ)$, $y \in (30^\circ, 50^\circ)$. Considerando sólo Z, encuentra los 4 días de 2021 más análogos al elemento a_0 := "2022/01/11" y calcula el error absoluto medio de la temperatura $\{T_{i,j,1000hPa}\}_{i,j}$, del subsistema, prevista para el elemento a_0 según la media de dichos análogos. Para la analogía, considera la distancia euclídea de elementos de σ con los pesos $w_{i,j,k} = w_{i,j}w_k$, donde $w_{i,j} = 1$ para las coordenadas (x, y) y $w_{500hPa} = w_{1000hPa} = 0{,}5$ y $w_k = 0$ para el resto de p_k.

Delimitando todos los sistemas como nos dice el enunciado, y considerando solo la altura geopotencial Z, los 4 días de 2021 más análogos al 11/01/2022 son los siguientes:

1. **23 / 03 / 2021** que está distancia 280.308
2. **16 / 01 / 2021**, que está a distancia 330.350
3. **12 / 01 / 2021**, que está a distancia 371.050
4. **16 / 03 / 2021**, que está a distancia 371.903

Estos días, como era de esperar, pertenecen a la misma estación que el día pedido, es decir, invierno. De hecho, la tercera fecha más análoga del año anterior difiere en un solo día con a_0. Esto es una forma de comprobar que nuestro algoritmo devuelve lo pedido de manera coherente, ya que los mismos días en años distintos suelen ser similares.

Con estos días, se ha calculado el error absoluto medio de la temperatura entre la predicha utilizando estos valores y la del día a_0, obteniendo el siguiente resultado:

Error Medio Absoluto de Temperatura (MAE) : 1.419 K (Kelvin)

Por lo tanto, la diferencia entre valor real y predicho es mínima, algo que se puede observar en la gráfica inferior derecha de la Ilustración 2 en comparación con los valores reales que aparecen en la gráfica inferior izquierda de la misma ilustración. En las gráficas superiores se observa los datos de altura geopotencial de ambas muestras, la de a_0 y la media de los días elegidos.

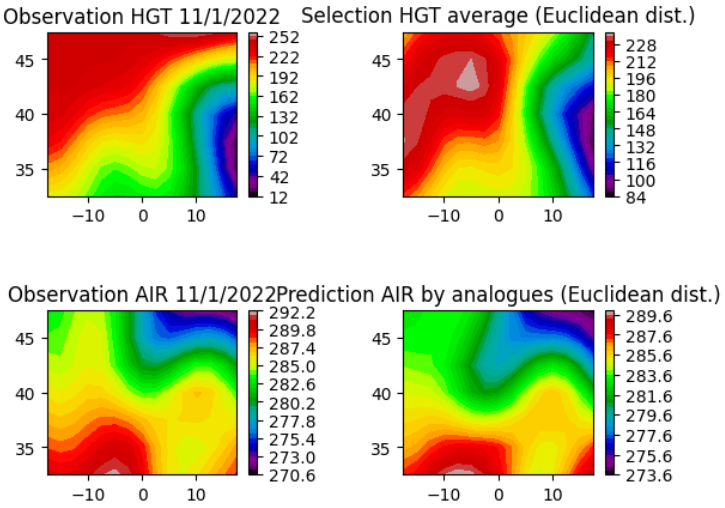

Ilustración 2

4. CONCLUSIÓN:

A través de esta práctica se logró realizar un análisis detallado de los cambios en la temperatura y la altura geopotencial de la atmósfera. Se identificaron las cuatro componentes principales, resaltando la importancia de la latitud a la hora del cálculo de la altura geopotencial, tal y como se ha visto en Resultados i) y en la gráfica PCA-1. Se determinaron los días más análogos al dado (a_0) basándose en la distancia euclídea, proporcionando un marco sólido para futuros análisis climatológicos. Los resultados aportados demuestran la eficiencia de este estudio, ya que la diferencia entre lo esperado y lo real es mínima. Este estudio también resaltó la importancia y la utilidad de los datos climatológicos para entender mejor los cambios en nuestro entorno.

5. **ANEXO:**

Código de Python utilizado:

```
"""
Author: PABLO FERNÁNDEZ DEL AMO
UNIT : GEOMETRÍA COMPUTACIONAL 2023

"""

import datetime as dt
import numpy as np
import matplotlib.pyplot as plt
from netCDF4 import Dataset
from sklearn.decomposition import PCA

def downsample_time(time, daily_samples=4):
    # Check that the length of the first axis is divisible by daily_samples
    assert len(time) % daily_samples == 0, "The length of the time axis must be divisible by
the number of daily samples"

    # Use advanced indexing to get every daily_samples-th element
    downsampled_time = time[::daily_samples]

    return downsampled_time

# Call the function on time

def daily_average(data, daily_samples=4):
    # Check that the length of the first axis is divisible by daily_samples
    assert data.shape[0] % daily_samples == 0, "The length of the time axis must be divisible
by the number of daily samples"

    # Reshape the vector so that the second axis is of size daily_samples
    reshaped_data = data.reshape(-1, daily_samples, *data.shape[1:])

    # Calculate the mean along the second axis
    daily_data = np.mean(reshaped_data, axis=1)

    return daily_data


# Load T (air) and Z (hgt) data for 2021 and 2022
def load_data():
    # Load air data for 2021

    air_data = Dataset("air.2021.nc", "r", format="NETCDF4")

    time_2021 = air_data.variables['time'][:].copy()
    pressure_level = air_data.variables['level'][:].copy()
    latitude = air_data.variables['lat'][:].copy()
    longitude = air_data.variables['lon'][:].copy()
    air_2021 = air_data.variables['air'][:].copy()
```

```
    air_data.close()

    # Load hgt data for 2021
    hgt_data = Dataset("hgt.2021.nc", "r", format="NETCDF4")

    hgt_2021 = hgt_data.variables['hgt'][:].copy()

    hgt_data.close()

    # Load air data for 2022
    air_data = Dataset("air.2022.nc", "r", format="NETCDF4")

    time_2022 = air_data.variables['time'][:].copy()
    air_2022 = air_data.variables['air'][:].copy()

    air_data.close()

    # Load hgt data for 2022
    hgt_data = Dataset("hgt.2022.nc", "r", format="NETCDF4")

    hgt_2022 = hgt_data.variables['hgt'][:].copy()

    hgt_data.close()

    air_2021 = daily_average(air_2021)
    air_2022 = daily_average(air_2022)

    hgt_2021 = daily_average(hgt_2021)
    hgt_2022 = daily_average(hgt_2022)

    time_2021 = downsample_time(time_2021)
    time_2022 = downsample_time(time_2022)

    return time_2021, pressure_level, latitude, longitude, air_2021, hgt_2021, time_2022,
air_2022, hgt_2022

# Adjust the longitude range so that Spain does not get split.
# We need the longitude to go from -180 to 180, and to do that, we need to rearrange the
# air_2021 and hgt_2021 data
def adjust_longitude_data(longitude, air_2021, hgt_2021, air_2022, hgt_2022):
    pos180 = np.argmax(longitude >= 180)
    air_2021 = np.roll(air_2021, axis = 3, shift = -pos180)
    hgt_2021 = np.roll(hgt_2021, axis = 3, shift = -pos180)
    air_2022 = np.roll(air_2022, axis = 3, shift = -pos180)
    hgt_2022 = np.roll(hgt_2022, axis = 3, shift = -pos180)
    longitude = np.roll(longitude, shift= -pos180)
    longitude[longitude >= 180] -= 360

    return longitude, air_2021, hgt_2021, air_2022, hgt_2022

# Estimate the 4 main components of the system setting p = 500 hPa
def apartadoI(time_2021, pressure_level, latitude, longitude, hgt_2021):
```

```
    n_components = 4
    hgt_2021b =
hgt_2021[:,pressure_level==500.,:,:].reshape(len(time_2021),len(latitude)*len(longitude))

    # Apply pca on the transpose matrix to reduce the number of elements (days)
    # and not the state variables
    X = hgt_2021b
    Y = hgt_2021b.transpose()
    pca = PCA(n_components= n_components)

    Element_pca0 = pca.fit_transform(Y)
    Element_pca0 =
Element_pca0.transpose(1,0).reshape(n_components,len(latitude),len(longitude))

    print(f"Explained variance ratio Y: {pca.explained_variance_ratio_}")

    pca.fit(X)
    print(f"Explained variance ratio X: {pca.explained_variance_ratio_}")

    fig = plt.figure()
    fig.subplots_adjust(hspace=0.4, wspace=0.4)

    for i in range(1, 5):
        ax = fig.add_subplot(2, 2, i)
        ax.text(0.5, 90, 'PCA-'+str(i), fontsize=18, ha='center')
        plt.contour(longitude, latitude, Element_pca0[i-1,:,:], cmap='nipy_spectral')
        plt.colorbar()

    fig = plt.gcf()
    fig.savefig("contours.pdf", format='png')
    plt.show()

# Considering the subsystem with x in (-20,20) and y in (30,50),
# Find the 4 most analogous days of 2021 to a0 = 2022/01/11 considering only hgt.
# Calculate the absolute mean error of the air (with p = 1000 hPa) expected
# for the element a0 according to the average of those analogues. For the analogy,
# we consider the Euclidean distance with weights 1 for x,y, 0.5 for the pressures
# of 500 and 1000 hPa and 0 for the rest.

def euclidean_distance(day0, day, pressure_level):

    day0_500 = 0.5*day0[pressure_level==500.,:,:]
    day_500 = 0.5*day[pressure_level==500.,:,:]

    day0_1000 = 0.5*day0[pressure_level==1000.,:,:]
    day_1000 = 0.5*day[pressure_level==1000.,:,:]
    diff500 = day0_500 - day_500
    diff1000 = day0_1000 - day_1000

    final_distance = np.sum((diff500).flatten()**2 + (diff1000).flatten()**2)

    final_distance = np.sqrt(final_distance)
```

```
    return final_distance


def apartadoII(time_2021, time_2022, pressure_level, latitude, longitude, air_2021, hgt_2021,
air_2022, hgt_2022):
    # Array of days changed to usual notation using the template provided by Robert
    dt_time21 = [dt.date(1800, 1, 1) + dt.timedelta(hours=t) for t in time_2021]
    dt_time22 = [dt.date(1800, 1, 1) + dt.timedelta(hours=t) for t in time_2022]

    # Identification fo the day a0 and its index in the array
    day0 = dt.date(2022, 1, 11)
    index_day0 = dt_time22.index(day0)


    # Restriction applied as stated in the second section
    air_restricted = air_2021[:,:,np.logical_and(latitude > 30, latitude < 50),:]
    air_restricted = air_restricted[:,:,:,np.logical_and(longitude > -20, longitude < 20)]
    hgt_restricted = hgt_2021[:,:,np.logical_and(latitude > 30, latitude < 50),:]
    hgt_restricted = hgt_restricted[:,:,:,np.logical_and(longitude > -20, longitude < 20)]

    # Data of temperature and geopotential height for a0 with restrictions


    air_a0_restricted = air_2022[:, :, np.logical_and(latitude > 30, latitude < 50),:]
    air_a0_restricted = air_a0_restricted[:, :, :,np.logical_and(longitude > -20, longitude <
20)]
    air_a0_restricted = air_a0_restricted[index_day0,:,:,:]

    hgt_a0_restricted = hgt_2022[:, :, np.logical_and(latitude > 30, latitude < 50),:]
    hgt_a0_restricted = hgt_a0_restricted[:, :, :, np.logical_and(longitude > -20, longitude
< 20)]
    hgt_a0_restricted = hgt_a0_restricted[index_day0,:,:,:]


    latitude_restricted = latitude[np.logical_and(latitude > 30, latitude < 50)]
    longitude_restricted = longitude[np.logical_and(longitude > -20, longitude < 20)]


    # Calculate the Euclidean distance between the reference day and every other day and
store the index of the day alongside its distance

    distance_index = [(euclidean_distance(hgt_a0_restricted,hgt_restricted[i, :, :, :],
pressure_level), i) for i in range(hgt_restricted.shape[0])]


    # Consider the first 4 days attending to the distance to a0
    days_number= 4
    distance_index.sort()

    # Find the day in the usual notation corresponding to the index stored in the distance-
index vector
```

```
    print(f"These are the {days_number} most analogous to
{str(day0.day)}/{str(day0.month)}/{str(day0.year)}:")
    index_analogues = []
    analogous_days = []
    counter = 0
    reps = 0
    # Print the most analogous days alongside its distance to a0, also append them ( if they
are different ) to analogous_days and their indexes to index_analogues.
    for _, index in distance_index:
        if dt_time21[index] not in analogous_days:
            index_analogues.append(index)
            analogous_days.append(dt_time21[index])
            counter += 1
            print(f"Day {counter} :
{str(dt_time21[index].day)}/{str(dt_time21[index].month)}/{str(dt_time21[index].year)} -
Distance = {distance_index[reps][0]}")
        reps += 1
        if counter == days_number:
            break




    hgt_average =
np.zeros((len(pressure_level),len(latitude_restricted),len(longitude_restricted)))
    air_average =
np.zeros((len(pressure_level),len(latitude_restricted),len(longitude_restricted)))

    for i in range(days_number):
        hgt_average += hgt_restricted[index_analogues[i]]
        air_average += air_restricted[index_analogues[i]]

    hgt_average /= days_number
    air_average /= days_number

    # Mean Absolute Error
    MAE = np.mean(abs(air_average[pressure_level==1000][0] -
air_a0_restricted[pressure_level==1000][0]))
    print(f"The mean absolute error (MAE) of the predicted {days_number} most analogous days
to {str(day0.day)}/{str(day0.month)}/{str(day0.year)} is:")
    print(MAE)

    # Plot the predictions and the real values of a0 using a similiar diagram to the one used
in section 1
    fig = plt.figure()
    fig.subplots_adjust(hspace=0.7, wspace=0.5)

    ax = fig.add_subplot(2, 2, 1)
    ax.title.set_text(f'Observation HGT {str(day0.day)}/{str(day0.month)}/{str(day0.year)}')
    plt.contourf(longitude_restricted, latitude_restricted, hgt_a0_restricted[pressure_level
== 1000,:,:][0], levels = 40, cmap='nipy_spectral')
    plt.colorbar()
```

```
    ax = fig.add_subplot(2, 2, 2)
    ax.title.set_text('Selection HGT average (Euclidean dist.)')
    plt.contourf(longitude_restricted, latitude_restricted, hgt_average[pressure_level ==
1000,:,:][0], levels = 40, cmap='nipy_spectral')
    plt.colorbar()

    ax = fig.add_subplot(2, 2, 3)
    ax.title.set_text(f'Observation AIR {str(day0.day)}/{str(day0.month)}/{str(day0.year)}')
    plt.contourf(longitude_restricted, latitude_restricted, air_a0_restricted[pressure_level
== 1000,:,:][0], levels = 40, cmap='nipy_spectral')
    plt.colorbar()

    ax = fig.add_subplot(2, 2, 4)
    ax.title.set_text('Prediction AIR by analogues (Euclidean dist.)')
    plt.contourf(longitude_restricted, latitude_restricted, air_average[pressure_level ==
1000,:,:][0], levels = 40, cmap='nipy_spectral')
    plt.colorbar()

    fig = plt.gcf()
    fig.savefig("ObservationvsPredicition.pdf", format='png')
    plt.show()

def main():
    time_2021, pressure_level, latitude, longitude, air_2021, hgt_2021,\
    time_2022, air_2022, hgt_2022 = load_data()
    longitude, air_2021, hgt_2021, air_2022, hgt_2022 = adjust_longitude_data(longitude,
air_2021, hgt_2021, air_2022, hgt_2022)
    apartadoI(time_2021, pressure_level, latitude, longitude, hgt_2021)
    apartadoII(time_2021, time_2022, pressure_level, latitude, longitude, air_2021, hgt_2021,
air_2022, hgt_2022)

if __name__ == '__main__':
    main()
```