UCL DEPARTMENT OF GEOGRAPHY

YEAR 2022-23

EXAM CANDIDATE ID:	**WQQR8**
MODULE CODE:	**GEOG0111**
MODULE NAME:	**Scientific Computing**
COURSE PAPER TITLE:	
WORD COUNT:	

Your essay, appropriately anonymised, may be used to help future students prepare for assessment. Double click this box to opt out of this ☐

Part B CW_V5

January 9, 2023

1 Geog0111 Coursework Part B by WQQR8

2 Part 1 SNOW DATA PREPARATION

[1]:
```python
def ttldoy(year:int):
    '''
    Get the total days of a year
    Take leap year into consideration
    It is more convinient than import a datetime pack
    '''

    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return 366
            else:
                return 365
        else:
            return 366
    else:
        return 365


def snowdata(year:int):
    '''
    Obtain satellite files form MODIS
    Read snow dataset from the files
    Read day of year (doy) from files
    Do smoothing and interpolation for snow cover data
    Output:
        interpolated snow cover data
        doy

    '''
    from geog0111.modisUtils import modisAnnual
    from osgeo import gdal
    import pandas as pd
```

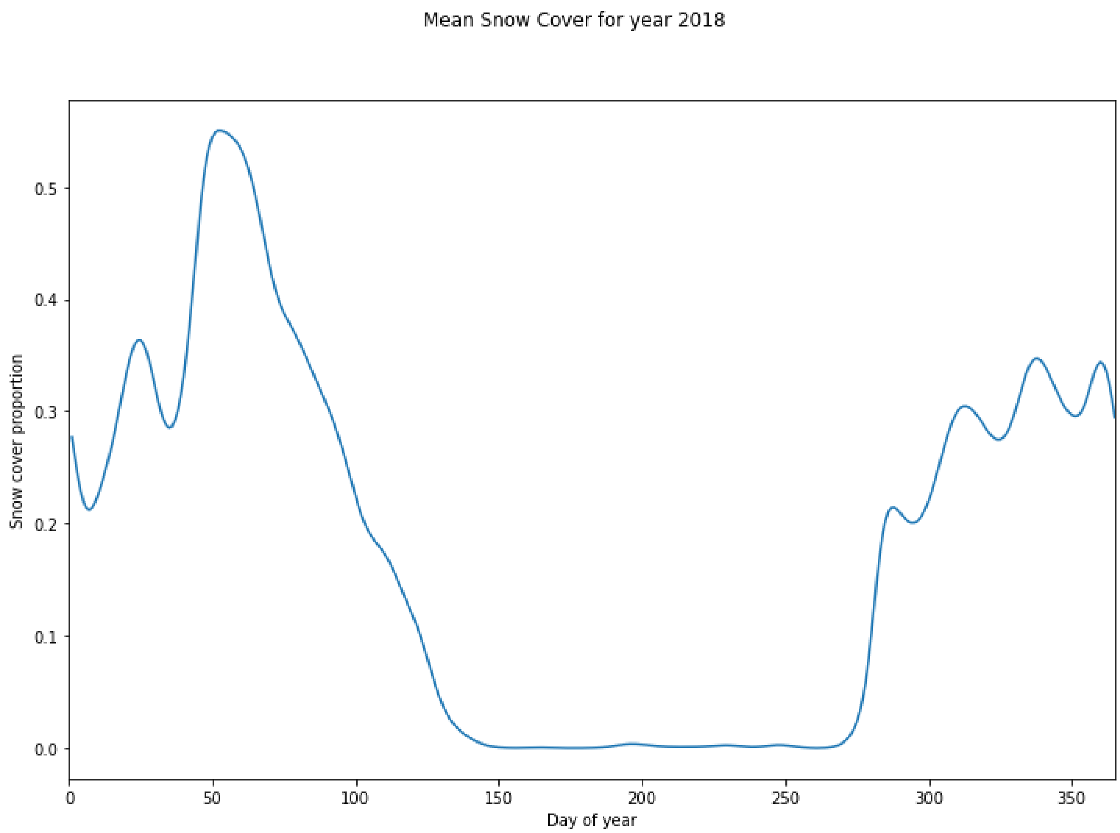
```python
import numpy as np
import scipy
import scipy.ndimage.filters

# Get total days of this year
t = ttldoy(year)

# Set up the argument of warp
warp_args = {
    'dstNodata'     : 255,
    'format'        : 'MEM',
    'cropToCutline' : True,
    'cutlineWhere'  : f"HUC=13010001",
    'cutlineDSName' : 'data/Hydrologic_Units/HUC_Polygons.shp'
}

kwargs = {
    'tile'      :    ['h09v05'],
    'product'   :    'MOD10A1',
    'sds'       :    ['NDSI_Snow_Cover'],
    'year'      :    year,
    'doys'      :    [i for i in range(1,t+1)],
    'warp_args' :    warp_args
}
# Obtain data form MODIS
filename,bandname = modisAnnual(verbose=False,**kwargs)

# Use gdal to read snow dataset
# Store the dataset as array in a dictionary
data_MOD10A1 = {}
for f,v in filename.items():
    g = gdal.Open(v)
    if g:
        data_MOD10A1[f] = g.ReadAsArray()

# Get snow cover data from snow dataset and scale it
snowcover= data_MOD10A1['NDSI_Snow_Cover'] * 0.01

# Get doy from dictionary
doy = np.array([int(i.split('-')[1]) for i in bandname])

# Determine the weight of snow cover data
# To prepare for the removel of non-snow data
weight = np.zeros_like(snowcover)
mask = (snowcover <= 1)
weight[mask] = 1
```

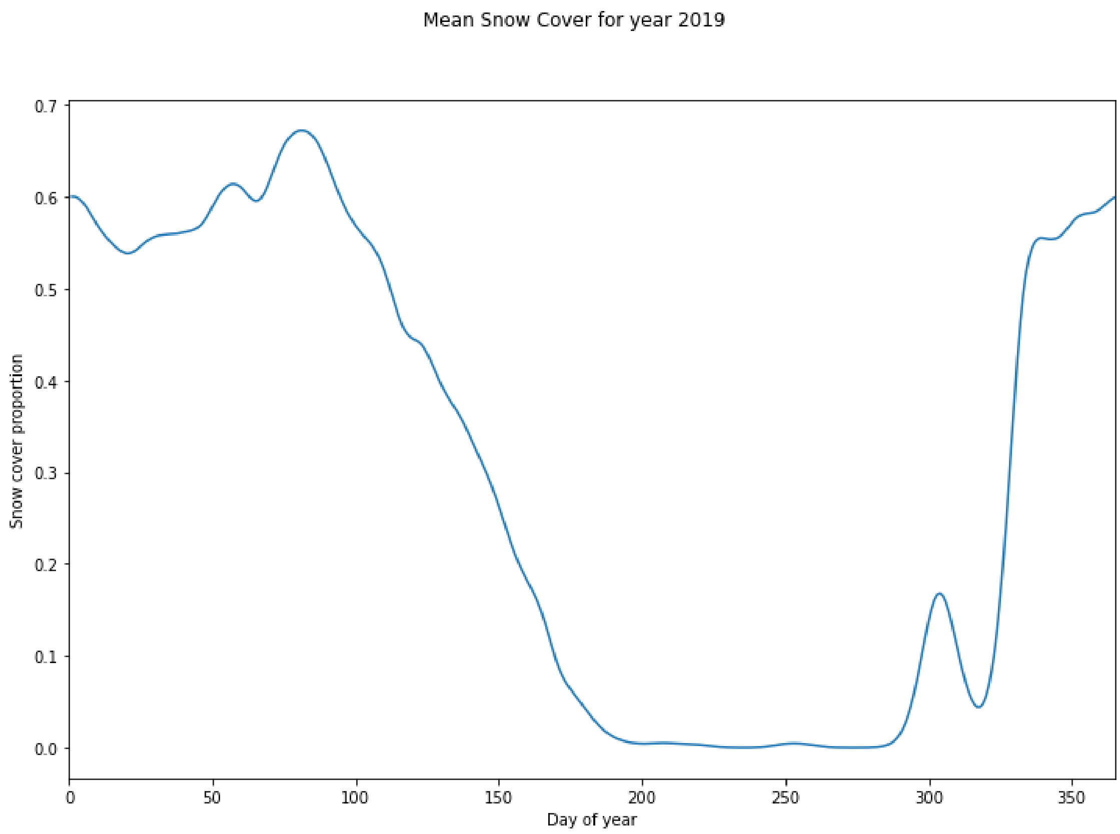
```
    # Define a gaussian distribution
    sigma = 5
    x = np.arange(-3*sigma,3*sigma+1)
    gaussian = np.exp((-(x/sigma)**2)/2.0)

    # Smoothing and interpolation by filter and convolution
    numerator = scipy.ndimage.filters.convolve1d(snowcover * weight, gaussian,␣
↪axis=0,mode='wrap')
    denominator = scipy.ndimage.filters.convolve1d(weight, gaussian,␣
↪axis=0,mode='wrap')

    # Avoid divide by 0 problems by setting zero values
    # of the denominator to not a number (NaN)
    denominator[denominator==0] = np.nan

    # Obtain the interpolated snow cover dataset
    snowcover_itpl = numerator/denominator

    # Output the interpolated snow cover data and doy
    return snowcover_itpl, doy
```

[2]:
```
def getsnowcover(year:int):
    '''
    Get snow cover dataset and doy
    Remove NaN in the dataset
    Calculate the mean of snow cover
    Write snow cover and doy into a pandas dataframe
    Store the dataframe in a csv format file
    Output:
        dataframe for plotting
    '''
    import numpy as np
    import pandas as pd
    from pathlib import Path

    # Get snow cover dataset and doy
    pdic, doy = snowdata(year)

    # Remove NaN in the dataset
    # Calculate the mean of snow cover
    mask = ~np.isnan(np.sum(pdic,axis = 0))
    mean_p =  np.mean(pdic[:,mask],axis=(1))

    # Write them into a pandas dataframe
    df = pd.DataFrame({'day_of_year':doy,'mean_snow_cover':mean_p})

    # Store the data in a csv format file
```

```
    df.to_csv(Path(f'work/snow_cover_{year}.csv'),index=False)

    # Output the dataframe for plotting
    return df
```

[3]:
```
import matplotlib.pyplot as plt
from scipy import signal

for year in [2018, 2019]:
    # Get dataframe includes mean snow cover and doy
    df = getsnowcover(year)

    # Set the plot size and type
    fig, axs = plt.subplots(1,1,figsize=(12,8))

    # Plot
    x = df['day_of_year']
    y = df['mean_snow_cover']
    axs.plot(x,y)

    # Set x-limits to get a neat graph
    d = list(df['day_of_year'])[-1]
    axs.set_xlim(0,d)

    # Set the plot title
    fig.suptitle(f'Mean Snow Cover for year {year}')

    # Set y-label
    axs.set_ylabel('Snow cover proportion')

    # Set x-label
    axs.set_xlabel('Day of year')
```


Mean Snow Cover for year 2018
Snow cover proportion
Day of year
0.0
0.1
0.2
0.3
0.4
0.5
0
50
100
150
200
250
300
350



3 Part 2 MODEL INVERSION

[4]:
```
# Import the model pack
from geog0111.model import*

def input_data(filepath):
    '''
    Read csv file as a dataframe
    Convert all NaN to numpy NaN, avoid influencing computation

    Output:
        dataframe
    '''

    import pandas as pd
    import numpy as np
    # Read csv file as a dataframe
    df = pd.read_csv(filepath)

    # Convert all NaN to numpy NaN, avoid influencing computation
```

```
    for value in df:
        if not isinstance(value, (int, float)):
            df.replace(value,np.nan,regex=True)

    return df

def RMSE(mod,obs):
    '''
    Calculate the root mean square error (RMSE) between modelled result and␣
 ↪observation

    Output:
        RMSE
    '''

    Qzip = zip(mod,obs)
    diffs = [x - y for x,y in Qzip]
    sq_diff = [diff**2 for diff in diffs]
    mse = sum(sq_diff)/len(sq_diff)
    rmse = mse**0.5

    return rmse
```

[5]:
```
def calibrate(year:int):
    '''
    Input the files with T and Q data to dataframe
    Input the files with p data to dataframe
    Get data T Q p t from dataframes
    Define the range of T0 and f in the lookup table
    Create an empty dictionary to store the lookup table values
    For each group of (T0,f), run the model to get modelled river flow
    Calculate the RMSE of modelled data and store it in the LUT
    Pick out the the parameter (T0,f) with best goodness of fit
    Store calibrated model parameters T0, f and the RMSE in dataframe
    Substitute calibrated parameters into model to get new modelled Q

    Parameters:
        T0 typically ranges from 0.0 to 20.0 ℃
        f  typically ranges from 5 to 20 days

    Output:
        dataframe (calibrated T0,f,RMSE)
        calibrated T0
        calibrated f
        modelled Q
        observed Q
        doy t
```

```
'''

from pathlib import Path
import pandas as pd

# Locate the files with T and Q data
TQfiles = Path('work',f'delNorte{year}.csv')
# Input the files with T and Q data, replace all NaN value
df_T_Q = input_data(TQfiles)

# Locate the files with p data
pfiles = Path('work',f'snow_cover_{year}.csv')
# Input the files with p data, replace all NaN value
df_p = input_data(pfiles)

# Get data T Q p t from dataframes
T = df_T_Q['mean_temperature']
Q = df_T_Q['stream_discharge']
p = df_p['mean_snow_cover']
t = df_T_Q['day_of_year']

# Define the range of T0 and f in the lookup table
min_T0, max_T0 = 0, 20
min_f, max_f = 5, 20

# Create an empty dictionary to store the lookup table values
LUT = {}

# Create parameter groups with all possible T0 and p
# For each group of (T0,f), run the model to get modelled river flow
# Calculate the RMSE of modelled data and store it in the LUT
for T0 in range(min_T0, max_T0 + 1):
    for f in range(min_f, max_f + 1):
        Q_mod_norm = model(T0,f,T,p).ravel()
        Q_obs_norm = Q/Q.sum(axis=0)
        z = RMSE(Q_mod_norm,Q_obs_norm)
        LUT[(T0,f)] = z

# Pick out the the parameter (T0,f) with best goodness of fit
best_fit = min(LUT.items(), key=lambda x: x[1])

# Unpack to get T0 and f
key, value = best_fit
T0_calib, f_calib = key

# Store calibrated model parameters T0, f and the RMSE in dataframe
```

```
    df = pd.DataFrame([T0_calib, f_calib, value],
                      index=['Calibrated_T0', 'Calibrated_f', 'RMSE'])
    # Substitute calibrated parameters into model to get new modelled Q
    Q_mod_calib_norm = model(T0_calib,f_calib,T,p).ravel()

    return df, T0_calib, f_calib, Q_mod_calib_norm, Q_obs_norm, t
```

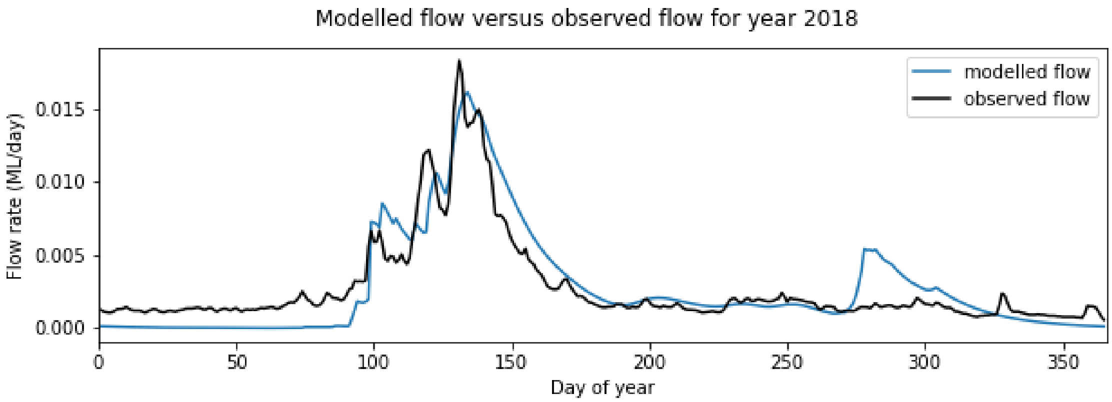
[6]:

```
# Run calibration function with data in 2018
calibration, T0, f, Q_mod_2018, Q_obs_2018, t= calibrate(2018)

print('The model parameters calibrated by year 2018')
print('(RMSE is applied to show the goodness of fit)')
print(calibration)
print('The RMSE between modelled flow and observed flow is very low,which␣
 ↪demostrates that the model fit the observation very well!')
print('But there is still a obvious offset in first 100 days, that can be␣
 ↪improved by further calibration.')
# Set plot size
fig, axs = plt.subplots(1,1,figsize=(10,3))

# Plot modelled Q for 2018 after calibration
axs.plot(t,Q_mod_2018,label='modelled flow')

# Plot observed Q
axs.plot(t,Q_obs_2018,'k',label='observed flow')

# Set limit of x-axis
axs.set_xlim(0,366)

# Set lengend
axs.legend(loc='best')

# Set the plot title
fig.suptitle(f'Modelled flow versus observed flow for year 2018')

# Set y-label
axs.set_ylabel('Flow rate (ML/day)')

# Set x-label
axs.set_xlabel('Day of year')
```

```
The model parameters calibrated by year 2018
(RMSE is applied to show the goodness of fit)
                       0
Calibrated_T0  12.000000
Calibrated_f   20.000000
RMSE            0.001504
```

```
The RMSE between modelled flow and observed flow is very low,which demostrates
that the model fit the observation very well!
But there is still a obvious offset in first 100 days, that can be improved by
further calibration.
```

[6]: Text(0.5, 0, 'Day of year')



[7]:
```python
def validate(year:int):
    '''
    Input the files with T and Q data to dataframe
    Input the files with p data to dataframe
    Get data T Q p t from dataframes
    Run the model
    Calculate the RMSE of modelled data
    Store RMSE in dataframe

    Output:
        dataframe (year, RMSE)
        modelled Q
        observed Q
        doy t
    '''

    from pathlib import Path
    import pandas as pd

    # Locate the files with T and Q data
    TQfiles = Path('work',f'delNorte{year}.csv')

    # Input the files with T and Q data, replace all NaN value
    df_T_Q = input_data(TQfiles)

    # Locate the files with p data
```

```
    pfiles = Path('work',f'snow_cover_{year}.csv')
    # Input the files with p data, replace all NaN value
    df_p = input_data(pfiles)

    # Get data T Q p t from dataframes
    T = df_T_Q['mean_temperature']
    Q = df_T_Q['stream_discharge']
    p = df_p['mean_snow_cover']
    t = df_T_Q['day_of_year']

    # Run the model
    Q_mod_norm = model(T0,f,T,p).ravel()
    Q_obs_norm = Q/Q.sum(axis=0)

    # Calculate the RMSE of modelled data
    rmse = RMSE(Q_mod_norm,Q_obs_norm)

    # Store RMSE in dataframe
    df = pd.DataFrame([year, rmse],
                      index=['Year', 'RMSE'])

    return df, Q_mod_norm, Q_obs_norm, t
```

[8]:

```
# Run validation function with data in 2019
result, Q_mod_2019, Q_obs_2019, t= validate(2019)

print('The model parameters validated by year 2019')
print('(RMSE is applied to show the goodness of fit)')
print('The RMSE between modelled flow and observed flow is very low,which␣
 ↪demostrates that the model fit the observation very well!')
print('The trend of the curve is basically the same!')
print(result)

# Set plot size
fig, axs = plt.subplots(1,1,figsize=(10,3))

# Plot modelled Q for 2018 after calibration
axs.plot(t,Q_mod_2019,label='modelled flow')

# Plot observed Q
axs.plot(t,Q_obs_2019,'k',label='observed flow')

# Set limit of x-axis
axs.set_xlim(0,366)

# Set lengend
axs.legend(loc='best')
```

```python
# Set the plot title
fig.suptitle(f'Modelled flow versus observed flow for year 2019')

# Set y-label
axs.set_ylabel('Flow rate (ML/day)')

# Set x-label
axs.set_xlabel('Day of year')
```

```
The model parameters validated by year 2019
(RMSE is applied to show the goodness of fit)
The RMSE between modelled flow and observed flow is very low,which demostrates
that the model fit the observation very well!
The trend of the curve is basically the same!
                0
Year  2019.000000
RMSE     0.001316
```

[8]: Text(0.5, 0, 'Day of year')