0000_partb_YMJR7

January 16, 2023

[1]:
```
from geog0111.modisUtils import modisAnnual
from osgeo import gdal
from geog0111.im_display import im_display
from geog0111.data_mask import data_mask
from geog0111.get_doy import get_doy
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os.path, time
import csv
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import scipy
import scipy.ndimage.filters

def getSnowData(year:int):
    '''
    Return a pandas dataframe, including the 'day_of_year' and 'snow_cover' of␣
 ↪a typical year we input
     '''
    #if the csv file exist, running a saved csv file directly will save time
    if os.path.exists(f'work/snow_cover_{year}.csv'):
        df = pd.read_csv(f'work/snow_cover_{year}.csv')
        #only return 'day_of_year','snow_cover'
        return df.iloc[:, [1, 2]]

    #define some necessary empty list
    snowCover = []
    doyList = []
    #std is used in regularisation
    std = []

    # define warp_args
    warp_args = {
    'dstNodata'     : 255,
```

```
'format' : 'MEM',
'cropToCutline' : True,
'cutlineWhere' : f"HUC=13010001",
'cutlineDSName' : 'data/Hydrologic_Units/HUC_Polygons.shp'
}
# define the number of days in each month
Day = {1:31,2:28,3:31,4:30,5:31,6:30,7:31,8:31,9:30,10:31,11:30,12:31,}
# use for loop to get snow cover dataset
for month in Day.keys():
    for day in range(1,Day[month]+1):
        #get doy
        doy = get_doy(year,month,day)
        doyList.append(doy)
        #define kwargs
        kwargs = {
                'tile'      :    ['h09v05'],
                'product'   :    'MOD10A1',
                'sds'       :    ['NDSI_Snow_Cover'],
                'year'      :    year,
                'doys'      :    [doy],
                'warp_args' :    warp_args
            }
        #use modisAnnual to get the data
        filename,bandname = modisAnnual(verbose=False,**kwargs)
        #define necessary parameters
        scale = [1]
        uthresh = [101]
        lthresh = [-1]
        sds = kwargs['sds']
        # read the data and save them in dataSaver temporarily
        dataSaver = {}
        for i,j in filename.items():
            g = gdal.Open(j)

            if g:
                dataSaver[i] = g.ReadAsArray()
        #use data_mask to mask the dataset
        dataSaver = data_mask(dataSaver,sds,scale,uthresh,lthresh)
        #calculate std and mean of snow cover by ingnoring NaN value
        snowCover.append(np.nanmean(dataSaver['NDSI_Snow_Cover']*0.01))
        std.append(np.nanstd(dataSaver['NDSI_Snow_Cover']))

# gap-filling
#find the NaN value
mask1 = np.isnan(snowCover)
mask2 = np.isnan(std)
# use for loop to replace NaN value to a previous one in snowCover
```

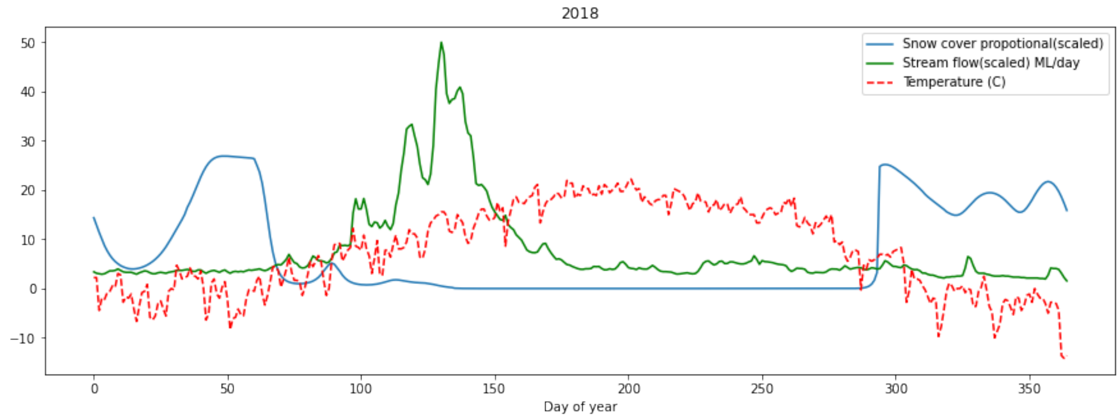
```python
    if mask1[0]:
        snowCover[0] = snowCover[1]
    for i in range(1, len(snowCover)):
        if mask1[i]:
            snowCover[i] = snowCover[i-1]
    # use for loop to replace NaN value to the mean of the first two elements␣
↪in std
    for i in range(1, len(std)):
        if mask2[i]:
            std[i] = (std[i-1]+std[i-2])/2

    #use get_weight() and regularise() to smooth the data
    weight           = get_weight(snowCover,std)
    interpolated_snow = regularise(snowCover,weight)
    mask             = make_mask(interpolated_snow)

    # mean over axis 1
    mean_snowCover = np.nanmean(interpolated_snow[:,mask],axis=(1))

    #create a dataframe and use zip() to combine doyList, snowCover and std␣
↪into a 3-dimensional array as data.
    df = pd.DataFrame(columns = ['day_of_year','snow_cover','std'],
    data =zip(doyList,mean_snowCover,std))
    #interpolation
    df['snow_cover'] = df['snow_cover'].interpolate()
    df['std'] = df['std'].interpolate()

    # save the file
    df.to_csv(f'work/snow_cover_{year}.csv')
    # setup Path object for output file
    filename = Path('work/snow_cover_'+str(year)+'.csv')
    # report
    print(f'file: {filename} written: {filename.stat().st_size} bytes')

    #show modification time of the file
    print("last modified: %s" % time.ctime(os.path.getmtime(filename)))

    # return the dataframe only return 'day_of_year','snow_cover'
    return df


#the following functions are use to smooth the data
#define weight
def get_weight(snowCover,std):
    weight = np.zeros_like(std)
    for i in range(1,len(std)):
        if std[i] == 0:
```

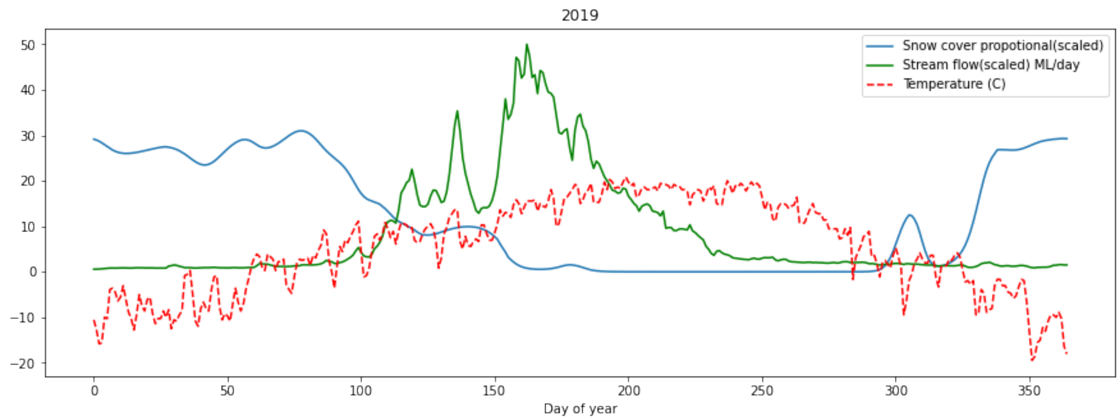
```
            std[i] = std[i-1]
        weight[i] = 1./(std[i]**2)

    return weight

# regularise
def regularise(snowCover,weight):
    ''' return regulaised dataset along axis 0'''
    sigma = 5
    x = np.arange(-3*sigma,3*sigma+1)
    gaussian = np.exp((-(x/sigma)**2)/2.0)

    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

    interpolated_snow = numerator/denominator
    return interpolated_snow

def make_mask(interpolated_snow):
    '''return True where there is no nan in axis 0'''
    return ~np.isnan(np.sum(interpolated_snow,axis=0))

# plot snow cover dataset with T and Q
def plotData(df, year:int):
    '''
    Purpose:
    Return a figure with snow cover, stream flow and the Temperature in a given␣
↪year by inputing a dataframe and year
    '''
    # define figure size
    plt.figure(figsize = (15,5))

    # define axis labels
    plt.xlabel('Day of year')


    # plot the data, *50 is used to scale
    plt.plot(df['snow_cover']*50, label = 'Snow cover propotional(scaled)')

    #plot T and Q
    # find and read the corresponding csv file
```

```python
    df2 = pd.read_csv(f'work/delNorte'+str(year)+'.csv')
    #scale
    Q = df2['stream discharge (ml/day)']
    Q = 50 * Q/Q.max()
    F = df2['average tempreture (Fahrenheit)']
    T = (F-32)*5/9

    plt.plot(Q, color = 'g',label= 'Stream flow(scaled) ML/day')
    plt.plot(T,'r--',label= 'Temperature (C)')
    #add the label of each lines
    plt.legend()
    # Figure tile
    plt.title(f'{year}')

# running help() for these function
help(getSnowData)

# define a function main() to call when a script
def main(year):
    # setup Path object for output file
    filename = Path('work/snow_cover_'+str(year)+'.csv')
    # report
    print(f'file: {filename} written: {filename.stat().st_size} bytes')
    #show modification time of the file
    '''
    The code in line203 is based on
    "How do I get file creation and modification date/times?"
    by Peter Mortensen and Bryan Oakley
    https://stackoverflow.com/questions/237079/
↪how-do-i-get-file-creation-and-modification-date-times
    '''
    print("last modified: %s" % time.ctime(os.path.getmtime(filename)))
    #print the plot
    print(plotData(getSnowData(year), year))

# calls main() if the file is run as a Python script
if __name__ == "__main__":
    main(2018)
    main(2019)
```

```
Help on function getSnowData in module __main__:

getSnowData(year: int)
    Return a pandas dataframe, including the 'day_of_year' and 'snow_cover' of a
typical year we input
```

```
file: work/snow_cover_2018.csv written: 17206 bytes
last modified: Sun Jan 15 18:44:44 2023
None
file: work/snow_cover_2019.csv written: 16898 bytes
last modified: Sun Jan 15 18:46:48 2023
None
```





[2]:
```
from scipy.special import expit
from geog0111.model import model
def calibrate(year):
    # find and read the corresponding csv file
    df1 = pd.read_csv(f'work/snow_cover_'+str(year)+'.csv')
    df2 = pd.read_csv(f'work/delNorte'+str(year)+'.csv')
    #scale
    p = df1['snow_cover']*50
    Q = df2['stream discharge (ml/day)']
    Q /= Q.sum()
    F = df2['average tempreture (Fahrenheit)']
```

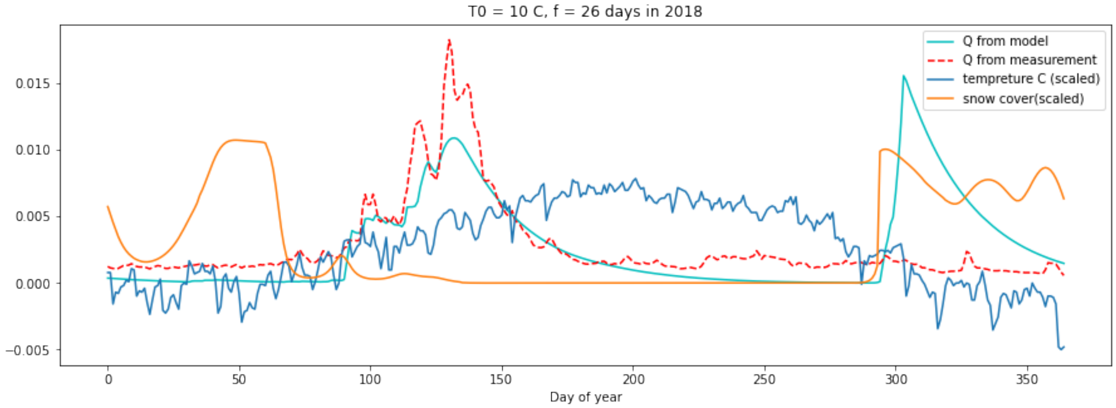
```
    T = (F-32)*5/9
    std = df1['std']
    #xp has a small impact, so assume xp = 1
    xp = 1
    #define measure_weight
    measure_weight = (1.96/std)**2

    #define TO list and f list
    TO = []
    f = []
    #use for loop to append the number
    for i in range(0,20,1):
        for j in range(5,31,1):
            f.append(j)
            TO.append(i)

    #define model list Q_model and rmse, using for loop to calculate their␣
 ↪values
    Q_model = []
    rmse = []
    #len(TO) and len(f) are 520
    for i in range(0,520,1):
        Qtem = model(TO[i],f[i],T,p).ravel()
        error = (Qtem - Q)*measure_weight
        rmse.append(np.sqrt(np.mean(error**2)))
        Q_model.append(Qtem)

    #create a dataframe including 'TO','f','rmse','Q_model'
    df = pd.DataFrame(columns = ['TO','f','rmse','Q_model'],
    data =zip(TO,f,rmse,Q_model))
    #interpolation
    df['TO'] = df['TO'].interpolate()
    df['f'] = df['f'].interpolate()
    df['rmse'] = df['rmse'].interpolate()
    df['Q_model'] = df['Q_model'].interpolate()

    # find the minimum RMSE and its index
    min_rmse = min(rmse)
    min_index = rmse.index(min_rmse)
    # find the corresponding TO and f
    TO_good= TO[min_index]
    f_good = f[min_index]
    #show the result
    print(f'TO = {TO_good} C, f = {f_good} days, rmse = {rmse[min_index]}')
    return df

def validate(year,calibration):
```

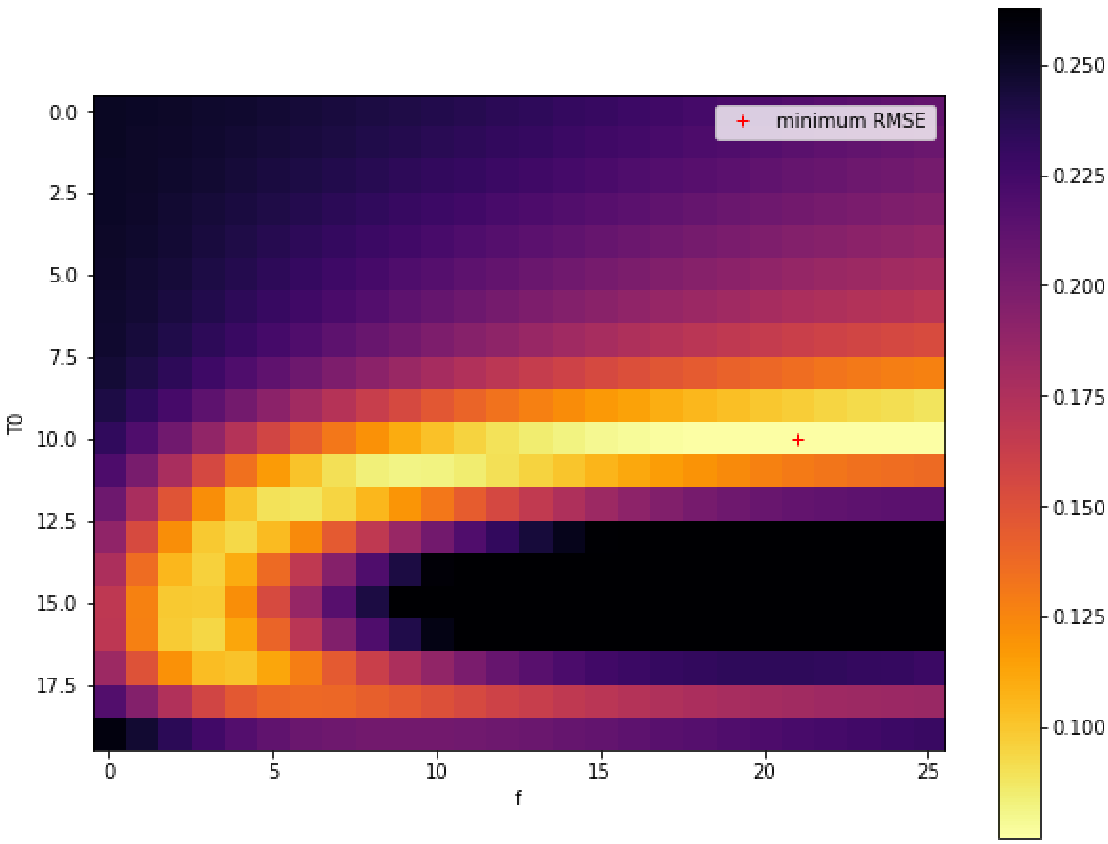
```python
'''
calibration should be calibrate(year)
'''
#get data
df = calibration
rmse = df['rmse'].tolist()
Q_model = df['Q_model']
T0 = df['T0'].tolist()
f = df['f'].tolist()
df1 = pd.read_csv(f'work/snow_cover_'+str(year)+'.csv')
df2 = pd.read_csv(f'work/delNorte'+str(year)+'.csv')
#scale
p = df1['snow_cover']*50
Q = df2['stream discharge (ml/day)']
Q /= Q.sum()
F = df2['average tempreture (Fahrenheit)']
T = (F-32)*5/9
doy = df1['day_of_year']
std = df1['std']

#reget min_index
min_rmse = min(rmse)
min_index = rmse.index(min_rmse)
# find the corresponding T0 and f
T0_good= T0[min_index]
f_good = f[min_index]

#plot time series plots of the modelled and measured flow data
# define figure size
plt.figure(figsize = (15,5))
p /= p.sum()
T /= T.sum()
plt.plot(Q_model[min_index], color = 'c', label= 'Q from model')
plt.plot(Q,'r--',label= 'Q from measurement')
plt.plot(T,label= 'tempreture C (scaled)')
plt.plot(p,label= 'snow cover(scaled)')
# define axis labels
plt.xlabel('Day of year')
#add the label of each lines
plt.legend()
# Figure tile
plt.title(f'T0 = {T0_good} C, f = {f_good} days in {year}')

#rmse visualisation
# Use mgrid as previously to define a 2D grid of parameters
T0min,T0max,T0step = 0.0,19.0,1
fmin,fmax,fstep = 5,30,1
```

```
    T0,f = np.mgrid[T0min:T0max+T0step:T0step,\
                    fmin:fmax+fstep:fstep]
    #decrease dimensions
    T0_ = np.ravel(T0)
    f_ = np.ravel(f)
    # min over time axis
    imin = np.argmin(rmse,axis=0)

    # back to 2D
    iT0min,ifmin = np.unravel_index(imin,T0.shape)
    T0min = T0[iT0min,ifmin]
    fmin = f[iT0min,ifmin]

    # plot Scatterplots
    rmse = np.array(rmse)
    fig, axs = plt.subplots(1,1,figsize=(10,8))
    im = axs.imshow(rmse.reshape(T0.shape),interpolation="nearest",\
                   vmax=min_rmse*3.5,cmap=plt.cm.inferno_r)
    fig.colorbar(im, ax=axs)
    axs.set_xlabel('f')
    axs.set_ylabel('T0')
    plt.plot([ifmin],[iT0min],'r+',label="minimum RMSE")
    axs.legend(loc='best')
    plt.show()

    #show the result
    if rmse[min_index] > 0.015:
        print('the parameters could be improved')
        return
    else:
        print('good match!')
        keys = ['T0','f', 'rmse']
        values = [T0_good, f_good, rmse[min_index]]
        result = dict(zip(keys, values))
        print(result)
    return result


if __name__ == "__main__":
    print(calibrate(2018))
    print(calibrate(2019))
    print('--'*30)
    validate(2018,calibrate(2018))
    validate(2019,calibrate(2019))
    validate(2019,calibrate(2018))
    msg = '''
    Ingoring rmse visualisation if there are two different year.
```

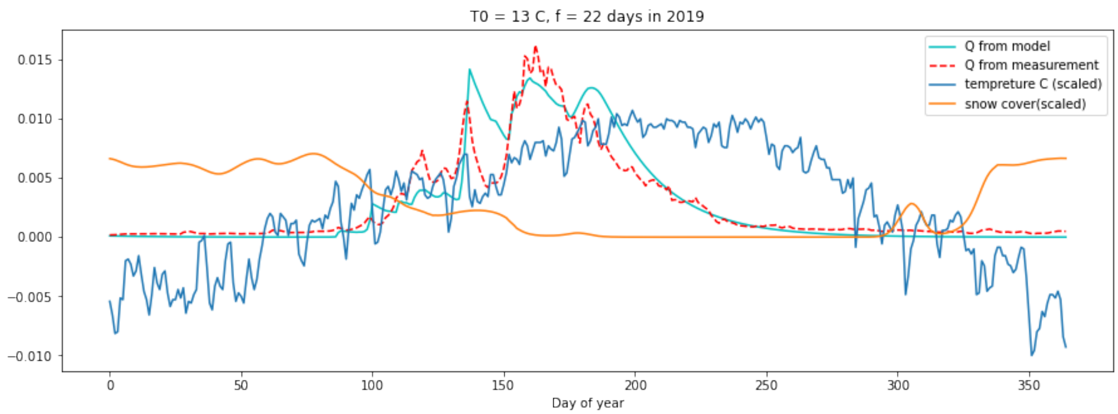
```
    Since my rmse is calculated in calibration and it is relevant to the␣
↪calibration input year.
    Maybe rmse should be calculated in the second function so we could get a␣
↪new rmse between
    model for the 2018 and the measurement of 2019
    '''
    print(msg)
```

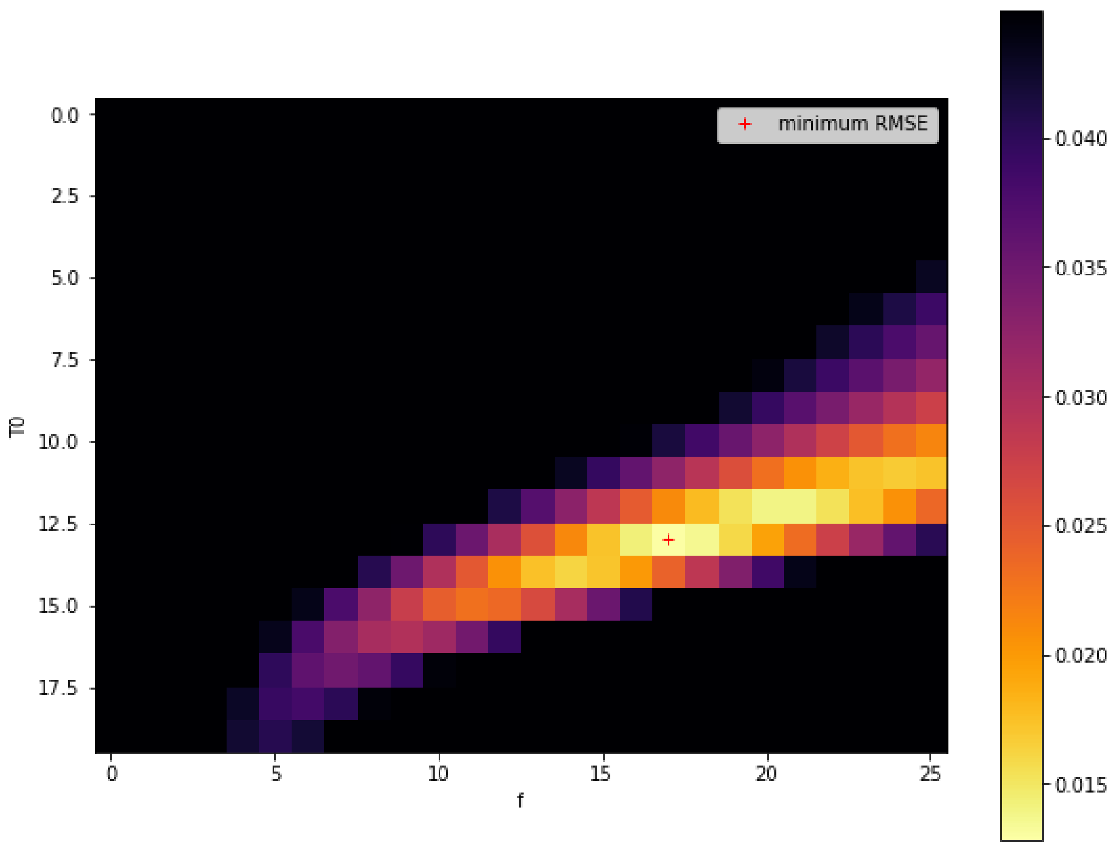
```
T0 = 10 C, f = 26 days, rmse = 0.0750921062395623
     T0   f      rmse                                            Q_model
0     0   5  0.250559  [0.004754785357074881, 0.006039522198217773, 0…
1     0   6  0.249864  [0.004358362017842359, 0.005506630120611376, 0…
2     0   7  0.248975  [0.004086427669908382, 0.0051178601396265145, …
3     0   8  0.247930  [0.003906566411360479, 0.00483763951475712, 0…
4     0   9  0.246764  [0.003795270962389074, 0.004639731676150363, 0…
..   ..  ..       …                                                 …
515  19  26  0.222028  [8.664971057299926e-05, 8.263813754500211e-05,…
516  19  27  0.223737  [0.00010063602636160665, 9.604464283492019e-05…
517  19  28  0.225366  [0.00011554250499007198, 0.0001103396492861334…
518  19  29  0.226911  [0.00013130117291699258, 0.0001254574832924580…
519  19  30  0.228373  [0.00014784123037748224, 0.0001413296759627225…

[520 rows x 4 columns]
T0 = 13 C, f = 22 days, rmse = 0.012836911435677499
     T0   f      rmse                                            Q_model
0     0   5  0.078056  [6.562962381897992e-05, 5.3747193016601324e-05…
1     0   6  0.077975  [7.906583188287292e-05, 6.693011965267439e-05,…
2     0   7  0.077856  [9.482778734314393e-05, 8.218918701895667e-05,…
3     0   8  0.077691  [0.00011522599464582653, 0.0001016576869698668…
4     0   9  0.077469  [0.0001420797078525106, 0.00012710567641893966…
..   ..  ..       …                                                 …
515  19  26  0.147228  [4.425399806522019e-05, 3.6970288724533384e-05…
516  19  27  0.152928  [5.3644408447240924e-05, 4.468573535006422e-05…
517  19  28  0.158324  [6.413247779626414e-05, 5.328144864370453e-05,…
518  19  29  0.163422  [7.570791736337524e-05, 6.274516025346535e-05,…
519  19  30  0.168232  [8.835799706072221e-05, 7.30645375895788e-05, …

[520 rows x 4 columns]
------------------------------------------------------------
T0 = 10 C, f = 26 days, rmse = 0.0750921062395623
```



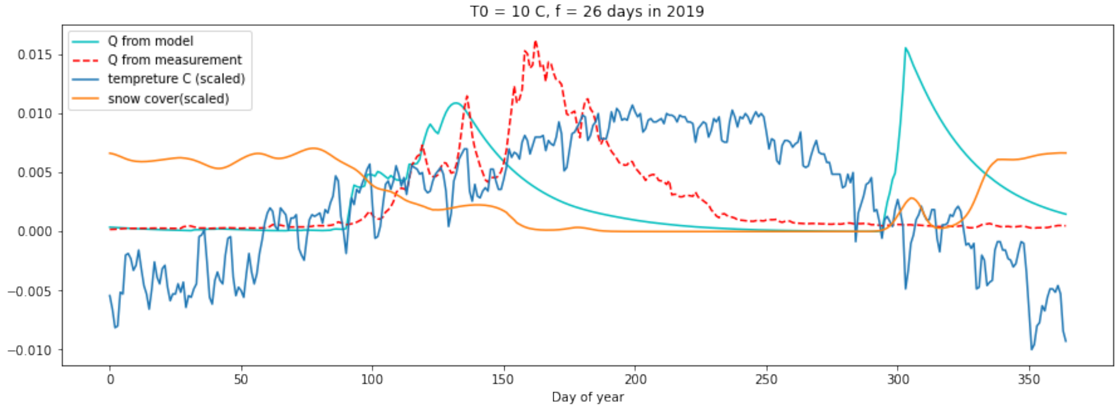


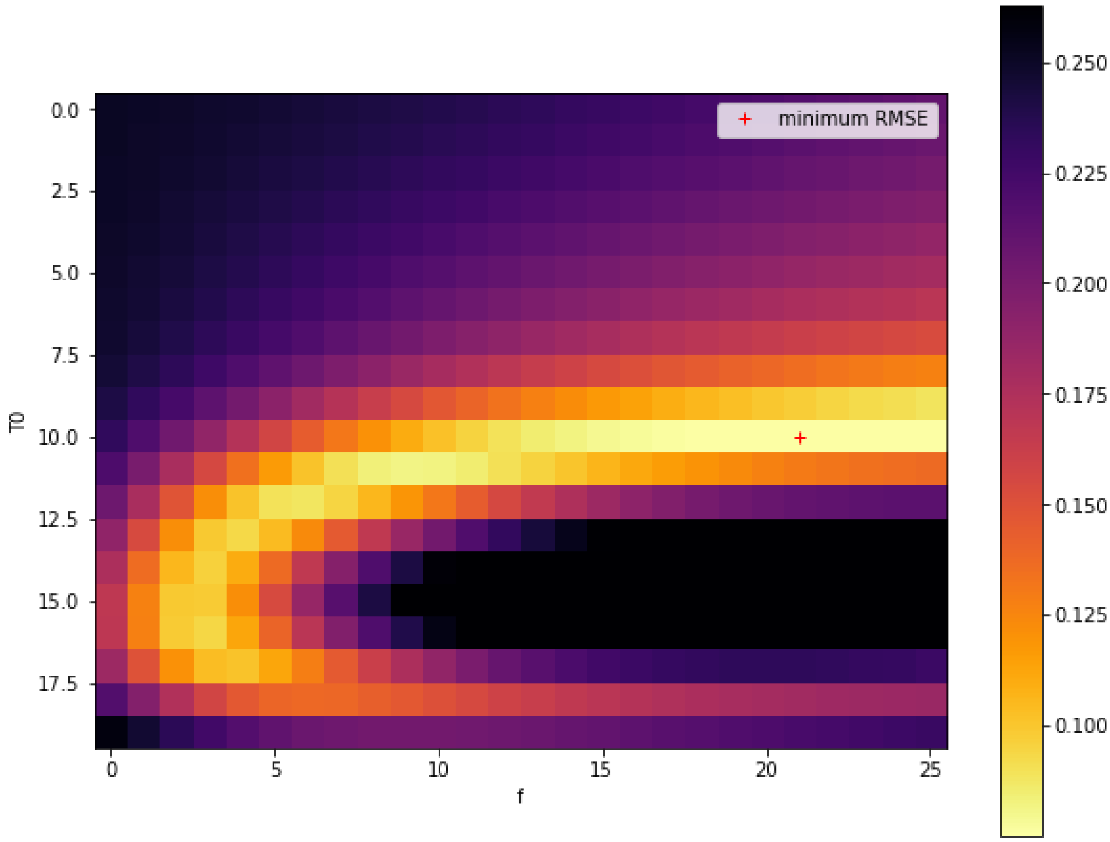

```
the parameters could be improved
T0 = 13 C, f = 22 days, rmse = 0.012836911435677499
```






```
good match!
{'T0': 13, 'f': 22, 'rmse': 0.012836911435677499}
T0 = 10 C, f = 26 days, rmse = 0.0750921062395623
```





the parameters could be improved

Ingoring rmse visualisation if there are two different year.
Since my rmse is calculated in calibration and it is relevant to the calibration input year.
Maybe rmse should be calculated in the second function so we could get a new

```
rmse between
    model for the 2018 and the measurement of 2019
```

[ ]:

```

```