SUEWS Parameters

Release 2020.1

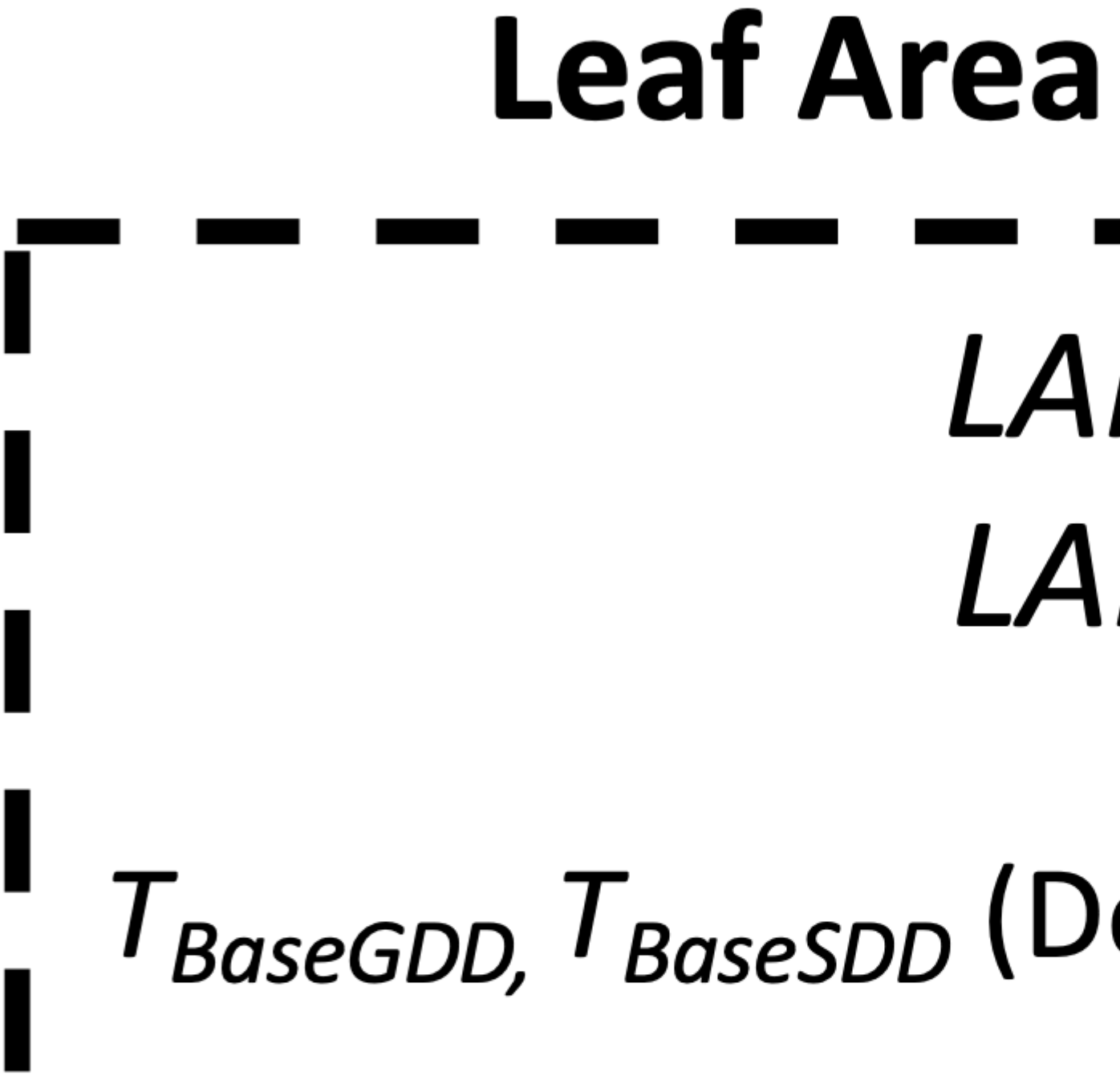
Jul 17, 2020

Overview

CHAPTER 1

Steps Overview

This is a tutorial to calculate various SUEWS parameters before running the model for the specific site and vegetation type. The parameters discussed here are: LAI, albedo, surface conductances, surface roughness and zero displacement height. Figures below shows how the process of parameters derivation should be conducted:


Leaf Area
LA
LA
$T_{BaseGDD}$, $T_{BaseSDD}$ (D

For more details of the tutorials, refer to here

This tutorial is based on:

Omidvar, H., Sun, T., Grimmond, S., Bilesbach, D., Black, A., Chen, J., Duan, Z., Gao, Z., Iwata, H., and McFadden, J. P.: Surface [Urban] Energy and Water Balance Scheme (v2020a) in non-urban areas: developments, parameters and performance, Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2020-148, in review, 2020.

1.1 Data

The meteorological observations used from Ameriflux (https://ameriflux.lbl.gov/) data are air temperature, incoming shortwave radiation, upwelling shortwave radiation, station pressure, relative humidity, wind speed, precipitation, net all-wave radiation, sensible heat flux and evaporation flux.

CHAPTER 2

Leaf Area Index related parameters

Necessary packages for analysis:

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from pathlib import Path
import supy as sp
import os
from shutil import copyfile
import geopandas as gpd
import pickle
pd.options.mode.chained_assignment = None
from IPython.display import clear_output
import warnings
warnings.filterwarnings('ignore')
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[2]:
```
from alb_LAI_util import generate_array_dif, generate_array_same, modify_attr,func_
↪parse_date
```

2.1 Parameters needed to calcuate LAI for non-crop vegetation (see Background https://umep-workshop.readthedocs.io/en/latest/Parameters/CalcBG.html)

- $BaseT$ -
- $BaseTe$
- LAI_{MAX}

- LAI_{Min}

2.2 Necessary functions for analysis

2.2.1 Read in the data (including LAI data)

[3]:
```
def read_data(year, name):

    df = pd.read_csv('data/MODIS_LAI_AmeriFlux/statistics_Lai_500m-' + name +
                     '.csv')
    if name!='crop':
        df.columns = ['product'
                      ] + [i.split(' ')[1] for i in df.columns if i != 'product']

    df = df.filter(['modis_date', 'value_mean'])

    df_period = df[[i.startswith('A' + str(year)) for i in df.modis_date]]
    df_period.loc[:, 'DOY'] = [
        int(i.split('A' + str(year))[1]) for i in df_period.modis_date
    ]
    df_period = df_period.set_index('DOY')

    copyfile("./runs/data/" + name + "_" + str(year) + "_data_60.txt",
             "runs/run/input/Kc_2012_data_60.txt")
    df_forcing = pd.read_csv('runs/run' + '/Input/' + 'kc' + '_' + '2012' +
                             '_data_60.txt',
                             sep=' ',
                             parse_dates={'datetime': [0, 1, 2, 3]},
                             keep_date_col=True,
                             date_parser=func_parse_date)

    all_sites_info = pd.read_csv('site_info.csv')
    site_info = all_sites_info[all_sites_info['Site Id'] == name]
    df = pd.DataFrame({
        'Site': [name],
        'Latitude': [site_info['Latitude (degrees)']],
        'Longitude': [site_info['Longitude (degrees)']]
    })

    path_runcontrol = Path('runs/run' + '/') / 'RunControl.nml'
    df_state_init = sp.init_supy(path_runcontrol)

    df_state_init ,level= modify_attr(df_state_init, df, name)

    grid = df_state_init.index[0]
    df_forcing_run = sp.load_forcing_grid(path_runcontrol, grid)

    df_output, df_state_final = sp.run_supy(df_forcing_run,
                                            df_state_init,
                                            save_state=False)

    return df_output, df_state_final, df_state_init, df_period, grid, df_forcing_run,
↪level
```

2.2.2 Calculating some of the variables of the models (LAI,GDD,SDD,Tair)

[4]:
```
def calc_vars(df_output, grid, df_forcing_run,year,level):

    df_output_2 = df_output.loc[grid, :]
    df_output_2 = df_output_2[df_output_2.index.year >= year]

    lai_model = pd.DataFrame(df_output_2.SUEWS.LAI)
    a = lai_model.index.strftime('%j')
    lai_model['DOY'] = [int(b) for b in a]

    if(level==1):
        nameGDD='GDD_DecTr'
        nameSDD='SDD_DecTr'
    elif(level==0):
        nameGDD='GDD_EveTr'
        nameSDD='SDD_EveTr'
    if(level==2):
        nameGDD='GDD_Grass'
        nameSDD='SDD_Grass'

    GDD_model = pd.DataFrame(df_output_2.DailyState[nameGDD])
    a = GDD_model.index.strftime('%j')
    GDD_model['DOY'] = [int(b) for b in a]
    GDD_model = GDD_model.dropna()
    GDD_model = GDD_model.iloc[1:]

    SDD_model = pd.DataFrame(df_output_2.DailyState[nameSDD])
    print(SDD_model)
    a = SDD_model.index.strftime('%j')
    SDD_model['DOY'] = [int(b) for b in a]
    SDD_model = SDD_model.dropna()
    SDD_model = SDD_model.iloc[1:]

    Tair = df_forcing_run.Tair.resample('1d', closed='left',
                                        label='right').mean()
    Tair = pd.DataFrame(Tair)
    a = Tair.index.strftime('%j')
    Tair['DOY'] = [int(b) for b in a]

    return lai_model, GDD_model, SDD_model, Tair,nameGDD,nameSDD
```

2.2.3 Calculate and plot the LAI for the site being analysed. The data are saved to a pickle file.

[5]:
```
def LAI_tune(year,name):
    df_output,df_state_final,df_state_init,df_period,grid,df_forcing_run,level = read_
↪data(year,name)
    lai_model,GDD_model,SDD_model,Tair,nameGDD,nameSDD=calc_vars(df_output,grid,df_
↪forcing_run,year,level)
    clear_output()

    with open('outputs/LAI/'+name+'-'+str(year)+'-MODIS.pkl','wb') as f:
        pickle.dump(df_period, f)
    with open('outputs/LAI/'+name+'-'+str(year)+'-Model.pkl','wb') as f:
```

(continues on next page)

(continued from previous page)

```
        pickle.dump(lai_model, f)

    attrs=[
    df_state_init.loc[:,'laimin'].loc[grid][level],
    df_state_init.loc[:,'laimax'].loc[grid][level],
    df_state_init.loc[:,'basete'].loc[grid][level],
    df_state_init.loc[:,'baset'].loc[grid][level]
    ]

    with open('outputs/LAI/'+name+'-attrs.pkl','wb') as f:
            pickle.dump(attrs, f)

    plt.rcParams.update({'font.size': 15})
    fig,axs=plt.subplots(2,1,figsize=(20,10))
    ax=axs[0]
    ax.plot(lai_model.DOY,lai_model.LAI,color='b',label='SUEWS-LAI')
    df_period.value_mean.plot(color='r',label='MODIS-LAI',ax=ax)
    ax.set_ylabel('LAI')
    ax.legend()
    ax.set_title(name+'-'+str(year))
    ax.set_xlabel('')

    ax=axs[1]
    plt.scatter(Tair.DOY,Tair.Tair,color='k')



    try:
        max_y=30
        for gdd_day in [90,100,110,120,130,140,150,170,180,210,260]:
            a=GDD_model[GDD_model.DOY==gdd_day][nameGDD]
            plt.plot([gdd_day,gdd_day],[-15,max_y],color='r')
            plt.annotate(str(np.round(a.values[0],0)),(gdd_day-5,-14),color='r',
↪rotation=90)


        for sdd_day in [150,170,180,200,250,270,300,310,320,330,340,350,360]:
            a=SDD_model[SDD_model.DOY==sdd_day][nameSDD]
            plt.plot([sdd_day,sdd_day],[-15,max_y],color='b')
            plt.annotate(str(np.round(a.values[0],0)),(sdd_day-5,-14),color='b',
↪rotation=90)
    except:
        pass

    ax.plot([0,365],[df_state_init.basete.iloc[0][0],df_state_init.basete.iloc[0][0]],
↪label='BaseTe',color='k')
    ax.plot([0,365],[df_state_init.baset.iloc[0][0],df_state_init.baset.iloc[0][0]],'-
↪-k',label='BaseT')
    ax.legend()
    plt.ylabel('Tair (C)')
    plt.xlabel('DOY')
    ax.set_xlim(left=0,right=365)
```

2.2.4 Now evaluate the LAI parameters at other 'Test' sites. Calculate and plot LAI (and save data to pickle files)

[6]:
```
def LAI_test(years,name):

    plt.rcParams.update({'font.size': 12})
    fig,axs=plt.subplots(len(years),2,figsize=(20,13))

    counter=-1
    for year in years:
        print(name+'-'+str(year))
        counter=counter+1
        df_output,df_state_final,df_state_init,df_period,grid,df_forcing_run,level =␣
↪read_data(year,name)
        lai_model,GDD_model,SDD_model,Tair,nameGDD,nameSDD=calc_vars(df_output,grid,
↪df_forcing_run,year,level)
        clear_output()

        with open('outputs/LAI/'+name+'-'+str(year)+'-MODIS.pkl','wb') as f:
            pickle.dump(df_period, f)
        with open('outputs/LAI/'+name+'-'+str(year)+'-Model.pkl','wb') as f:
            pickle.dump(lai_model, f)


        ax=axs[counter][0]
        ax.plot(lai_model.DOY,lai_model.LAI,color='b',label='SUEWS-LAI')
        df_period.value_mean.plot(color='r',label='MODIS-LAI',ax=ax)
        ax.set_ylabel('LAI')
        if counter==0:
            ax.legend()
        if counter!=len(years)-1:
            ax.set_xlabel('')
        ax.set_title(name+'-'+str(year))




        ax=axs[counter][1]
        ax.scatter(Tair.DOY,Tair.Tair,color='k')
        try:
            max_y=30
            for gdd_day in [90,110,130,150]:
                a=GDD_model[GDD_model.DOY==gdd_day][nameGDD]
                ax.plot([gdd_day,gdd_day],[-15,max_y],color='r')
                ax.annotate(str(np.round(a.values[0],0)),(gdd_day-10,-14),color='r',
↪rotation=90)


            for sdd_day in [300,320,340,360]:
                a=SDD_model[SDD_model.DOY==sdd_day][nameSDD]
                ax.plot([sdd_day,sdd_day],[-15,max_y],color='b')
                ax.annotate(str(np.round(a.values[0],0)),(sdd_day-10,-14),color='b',
↪rotation=90)
        except:
            pass
        ax.plot([0,365],[df_state_init.basete.iloc[0][0],df_state_init.basete.
↪iloc[0][0]],label='BaseTe',color='k')
```

(continues on next page)

(continued from previous page)

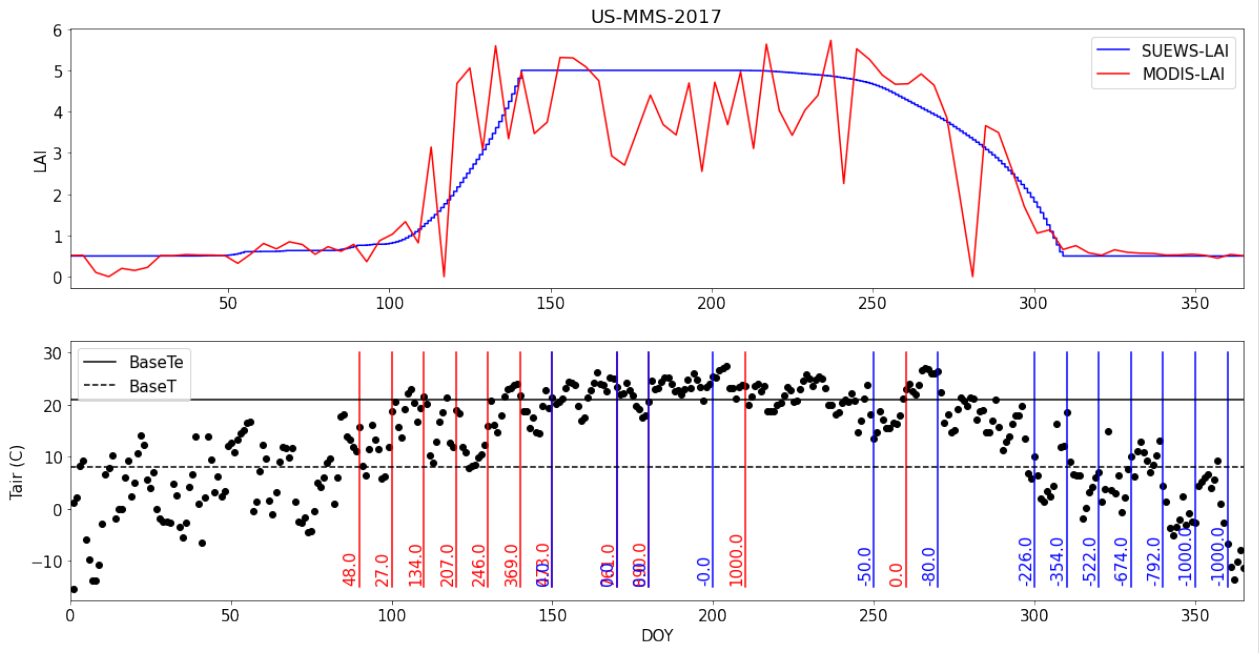
```
        ax.plot([0,365],[df_state_init.baset.iloc[0][0],df_state_init.baset.
↪iloc[0][0]],'--k',label='BaseT')
        if counter==0:
            ax.legend()
        ax.set_ylabel('Tair (C)')
        if counter==len(years)-1:
            ax.set_xlabel('DOY')
        ax.set_xlim(left=0,right=365)
        ax.set_title(name+'-'+str(year))

    plt.savefig('figs/'+name+'-LAI.png',dpi=300,bbox_inches = 'tight',pad_inches = 0.
↪01)
```

2.3 Plot for all sites

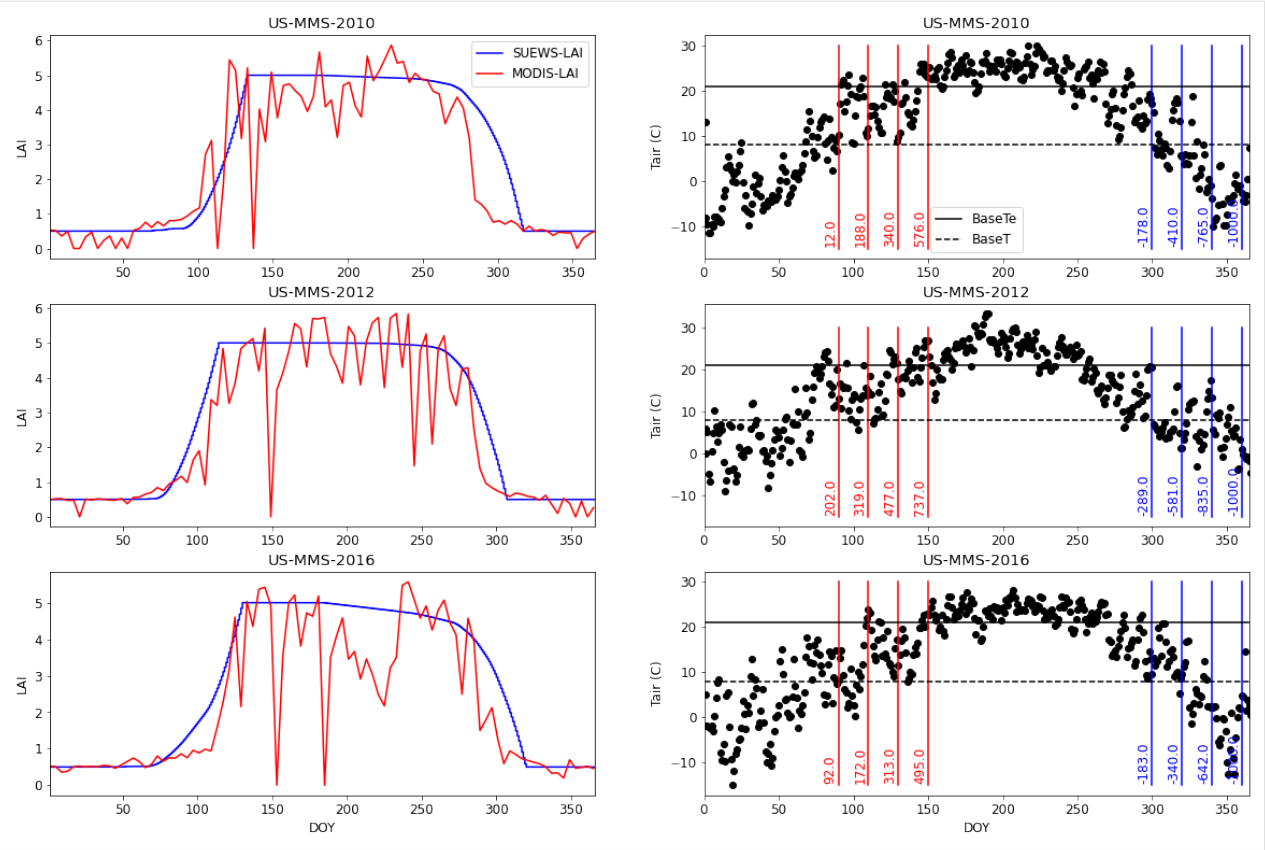
2.3.1 DecTr

US-MMS

[7]:
```
name='US-MMS'
year=2017
LAI_tune(year,name)
```



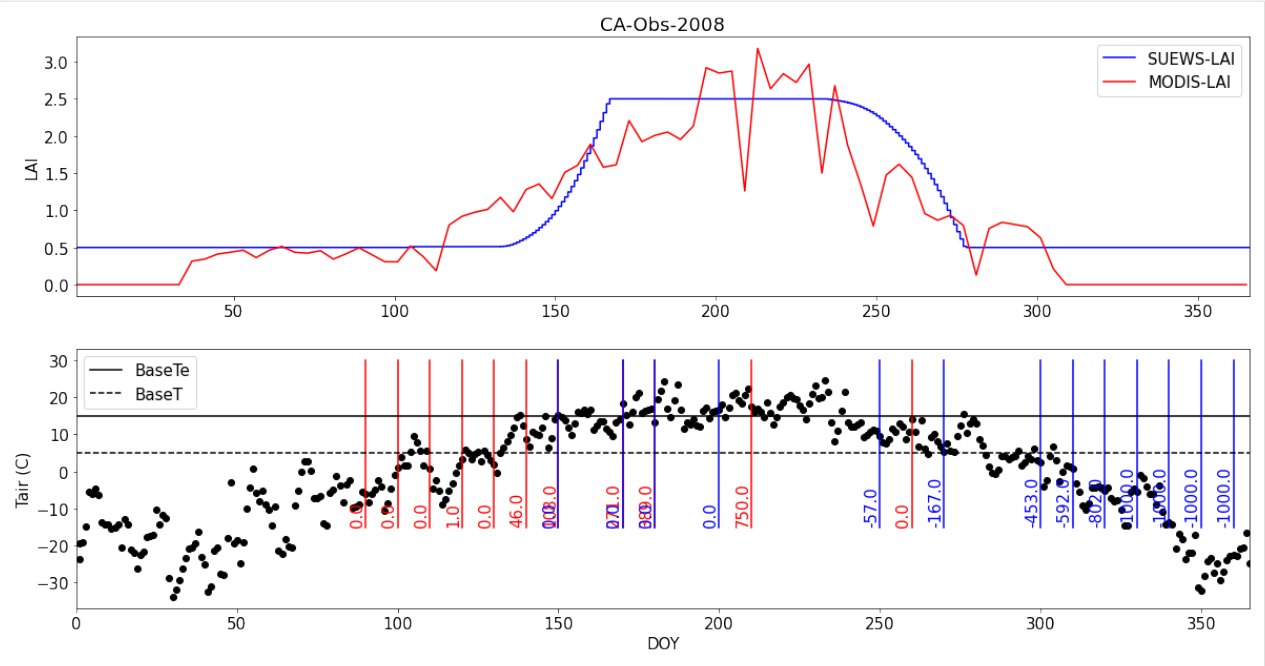
[8]:
```
name='US-MMS'
years=[2010,2012,2016]
LAI_test(years,name)
```



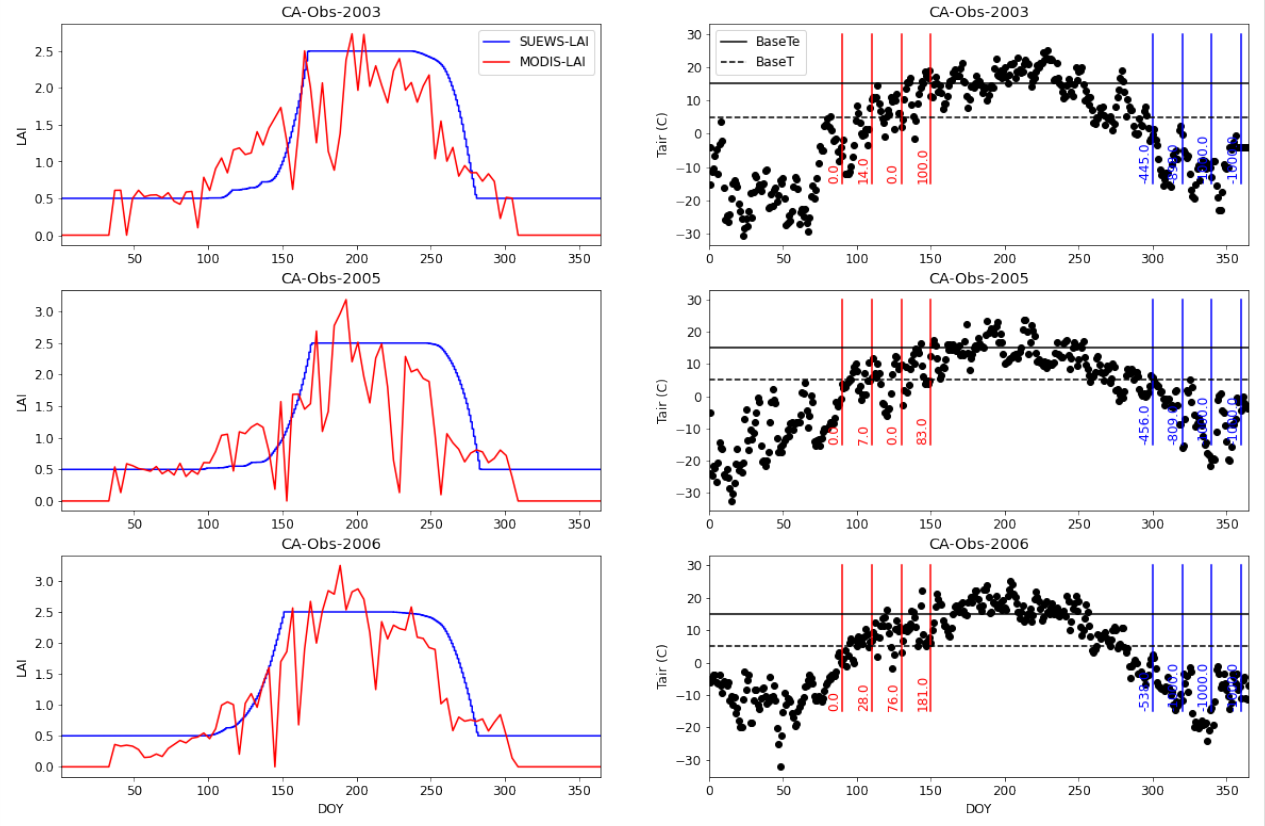
2.3.2 EveTr

CA-Obs

[9]:
```
name = 'CA-Obs'
year = 2008
LAI_tune(year, name)
```



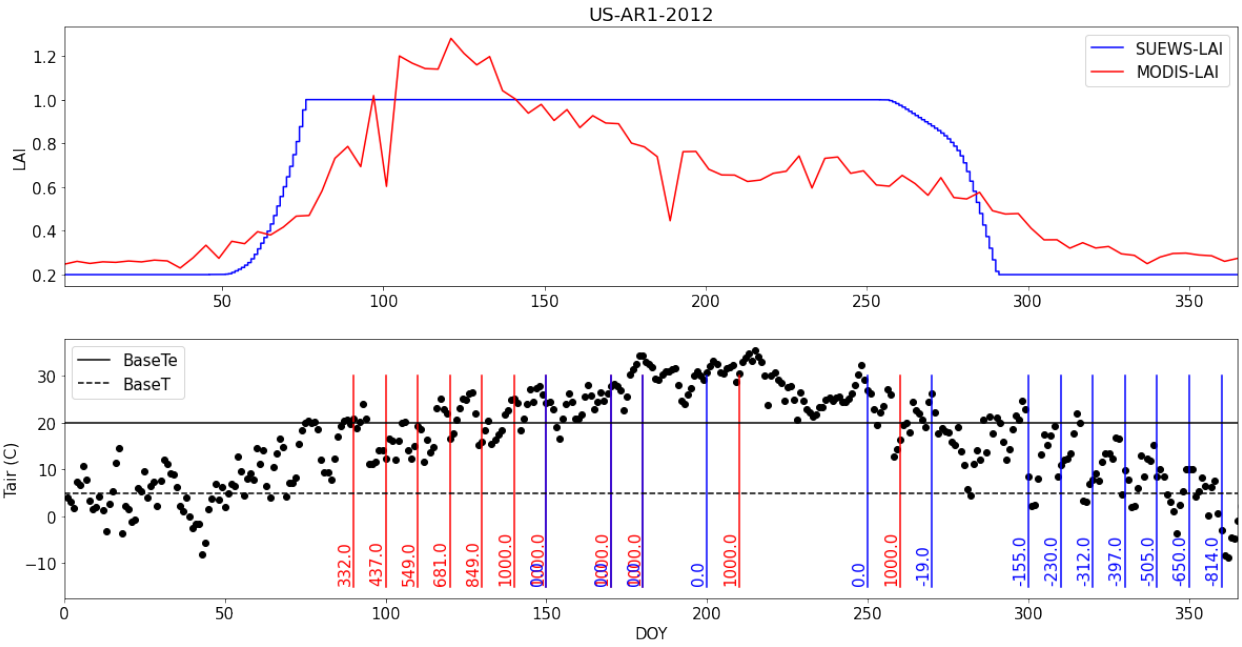
[10]:
```
name = 'CA-Obs'
years = [2003, 2005, 2006]
LAI_test(years, name)
```



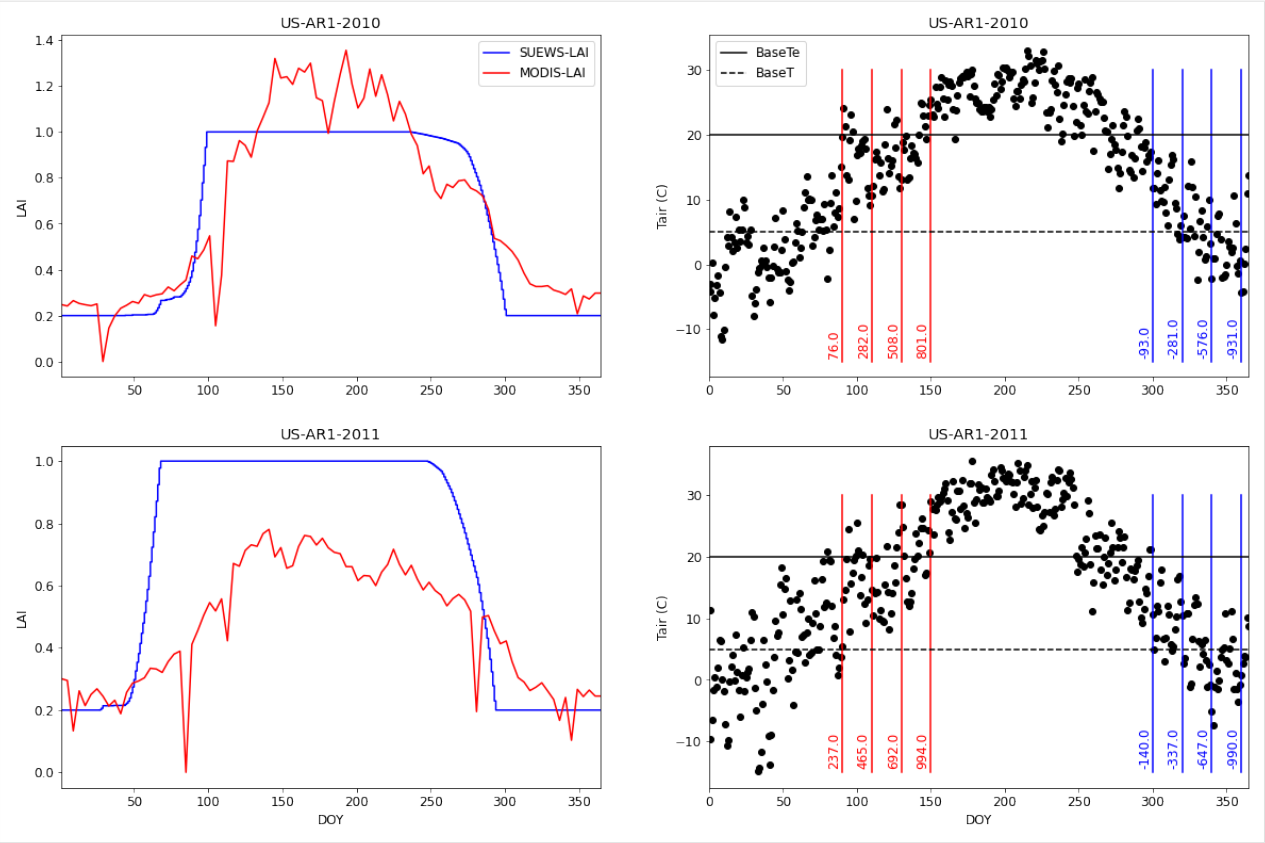
2.3.3 Grass

US-AR1

[11]:
```
name = 'US-AR1'
year = 2012
LAI_tune(year, name)
```



[12]:
```
name = 'US-AR1'
years = [2010, 2011]
LAI_test(years, name)
```


US-AR1-2010
SUEWS-LAI
MODIS-LAI
LAI
US-AR1-2010
BaseTe
BaseT
Tair (C)
76.0
282.0
508.0
801.0
-93.0
-281.0
-576.0
-931.0
US-AR1-2011
LAI
DOY
US-AR1-2011
Tair (C)
237.0
465.0
692.0
994.0
-140.0
-337.0
-647.0
-990.0
DOY

CHAPTER 3

Albedo parameters

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
from pathlib import Path
import supy as sp
import os
from shutil import copyfile
import geopandas as gpd
import pickle
pd.options.mode.chained_assignment = None
from IPython.display import clear_output
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[3]:
```
from alb_LAI_util import generate_array_dif, generate_array_same, modify_attr,func_
↪parse_date
```

3.1 Parameters to tune for α

- $\alpha_{LAI_{max}}$
- $\alpha_{LAI_{min}}$

3.2 Necessary functions for analysis

3.2.1 Reading data, calculating α and plotting gainst the observation (saving to pickle files)

[26]:
```
def read_plot(years,name,multiple_run=0):

    for year in years:
        print(name+'-'+str(year))
        df=pd.read_csv('data/data_csv_zip_clean/'+name+'_clean.csv.gz')
        df.time=pd.to_datetime(df.time)
        df=df.set_index(['time'])


        period_start=str(year)+'-01-01'
        period_end=str(year+1)+'-01-01'
        df_period=df[(df.index>=period_start) & (df.index<period_end)]



        df_period=df_period[df_period.SWIN>5]
        df_period=df_period[(df_period.index.hour <=14) & (df_period.index.hour >=10)]
        alb_raw=df_period['SWOUT']/df_period['SWIN']
        alb_raw=alb_raw.resample('1D').mean()
        alb=alb_raw[(alb_raw>0) & (alb_raw<1)]


        alb_avg_day=pd.DataFrame(alb,columns=['alb'])



        a=alb_avg_day.index.strftime('%j')
        alb_avg_day['DOY']=[int(b) for b in a]


        copyfile("./runs/data/"+name+"_"+str(year)+"_data_60.txt", "runs/run/input/Kc_
↪2012_data_60.txt")
        df_forcing=pd.read_csv('runs/run'+'/Input/'+'kc'+'_'+'2012'+'_data_60.txt',
↪sep=' ',
                                        parse_dates={'datetime': [0, 1, 2, 3]},
                                        keep_date_col=True,
                                        date_parser=func_parse_date)



        df_forcing.loc[:,'snow']=.8
        rol=df_forcing.Tair.rolling(5).mean()
        snowdetected=1
        for i in range(len(df_forcing)):

            if snowdetected==1:
                if rol.iloc[i]>=5:
                    df_forcing.loc[df_forcing.iloc[i].name,'snow']=0
                    snowdetected=0
                else:
                    df_forcing.loc[df_forcing.iloc[i].name,'snow']=0.8
```

(continues on next page)

(continued from previous page)

```
            else:
                df_forcing.loc[df_forcing.iloc[i].name,'snow']=0


            if (df_forcing.iloc[i].Tair<0) and (df_forcing.iloc[i].rain>0):
                df_forcing.loc[df_forcing.iloc[i].name,'snow']=0.8
                snowdetected=1


        all_sites_info =  pd.read_csv('site_info.csv')
        site_info=all_sites_info[all_sites_info['Site Id'] == name]
        df = pd.DataFrame(
            {'Site': [name],
            'Latitude': [site_info['Latitude (degrees)']],
            'Longitude': [site_info['Longitude (degrees)']]})



        path_runcontrol = Path('runs/run'+'/') / 'RunControl.nml'
        df_state_init = sp.init_supy(path_runcontrol)


        df_state_init,level=modify_attr(df_state_init, df, name)

        if level==1:
            attrs=[
                df_state_init.albmin_dectr,
                df_state_init.albmax_dectr
                ]
        elif level==0:
            attrs=[
                df_state_init.albmin_evetr,
                df_state_init.albmax_evetr
                ]
        elif level ==2:
            attrs=[
                df_state_init.albmin_grass,
                df_state_init.albmax_grass
                ]

        with open('outputs/albedo/'+name+'-attrs_albedo.pkl','wb') as f:
            pickle.dump(attrs, f)

        grid = df_state_init.index[0]
        df_forcing_run = sp.load_forcing_grid(path_runcontrol, grid)

        if multiple_run == 0:
            df_output, df_state_final = sp.run_supy(df_forcing_run, df_state_init,␣
↪save_state=False)

        if multiple_run == 1:
            error=20
            for i in range(10):

                if (error <= 0.1):
                    break
                df_output, df_state_final = sp.run_supy(df_forcing_run, df_state_init,
↪save_state=False)
```

(continues on next page)

(continued from previous page)

```
            final_state = df_state_final[df_state_init.columns.levels[0]].iloc[1]
            df_state_init.iloc[0] = final_state
            soilstore_before = df_state_final.soilstore_id.iloc[0]
            soilstore_after = df_state_final.soilstore_id.iloc[1]
            diff_soil = sum(abs(soilstore_after-soilstore_before))
            error = 100*diff_soil/soilstore_before.mean()
            print(error)

    df_output_2=df_output.SUEWS.loc[grid]
    df_output_2=df_output_2[df_output_2.index.year>=year]

    alb_model=pd.DataFrame(df_output_2.AlbBulk)
    a=alb_model.index.strftime('%j')
    alb_model['DOY']=[int(b) for b in a]


    Tair=df_forcing_run.Tair.resample('1d', closed='left',label='right').mean()
    Tair=pd.DataFrame(Tair)
    a=Tair.index.strftime('%j')
    Tair['DOY']=[int(b) for b in a]

    lai=df_output_2.LAI
    lai=lai.resample('1d', closed='left',label='right').mean()
    lai=pd.DataFrame(lai)
    a=lai.index.strftime('%j')
    lai['DOY']=[int(b) for b in a]

    snow=df_forcing.snow
    snow.index=df_forcing.datetime
    snow=snow.resample('1D').mean()
    snow=pd.DataFrame(snow)
    a=snow.index.strftime('%j')
    snow['DOY']=[int(b) for b in a]

    rain=df_forcing_run.rain
    rain=rain.resample('1d', closed='left',label='right').sum()
    rain=pd.DataFrame(rain)
    a=rain.index.strftime('%j')
    rain['DOY']=[int(b) for b in a]

    SMD=df_output_2.SMD
    SMD=SMD.resample('1d', closed='left',label='right').mean()
    SMD=pd.DataFrame(SMD)
    a=SMD.index.strftime('%j')
    SMD['DOY']=[int(b) for b in a]

    out={'obs':{'x':alb_avg_day.DOY,'y':alb_avg_day.alb},
         'model':{'x':alb_model.DOY,'y':alb_model.AlbBulk},
         'Tair':{'x':Tair.DOY,'y':Tair.Tair},
         'lai':{'x':lai.DOY,'y':lai.LAI},
         'snow':{'x':snow.DOY,'y':snow.snow},
         'rain':{'x':rain.DOY,'y':rain.rain},
         'smd':{'x':SMD.DOY,'y':SMD.SMD},
         }
    with open('outputs/albedo/'+name+'-'+str(year)+'-all-albedo.pkl','wb') as f:
        pickle.dump(out, f)
```

(continues on next page)

(continued from previous page)

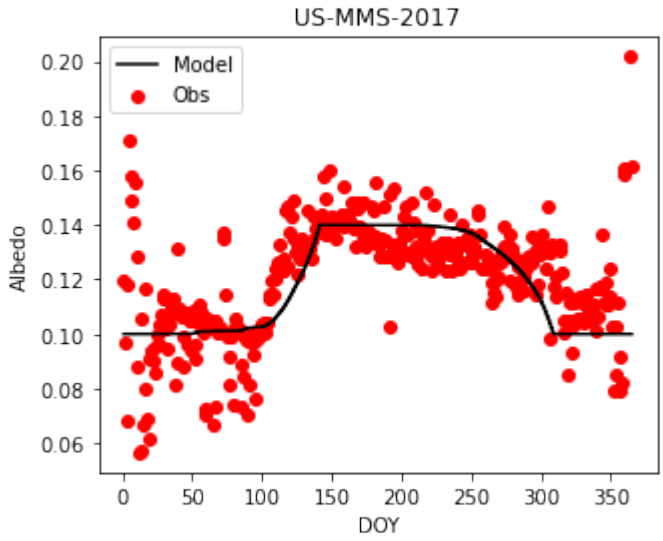
```
        clear_output()
    fig,axs=plt.subplots(len(years),1,figsize=(5,4*len(years)))
    plt.subplots_adjust(hspace=0.3)
    counter=-1
    for year in years:
        counter += 1
        try:
            ax=axs[counter]
        except:
            ax=axs
        with open('outputs/albedo/'+name+'-'+str(year)+'-all-albedo.pkl','rb') as f:
            out=pickle.load(f)
        ax.scatter(out['obs']['x'],out['obs']['y'],color='r',label='Obs')
        ax.plot(out['model']['x'],out['model']['y'],color='k',label='Model')
        ax.legend()
        ax.set_title(name+'-'+str(year))
        ax.set_xlabel('DOY')
        ax.set_ylabel('Albedo')
```

3.3 Plotting for all the sites

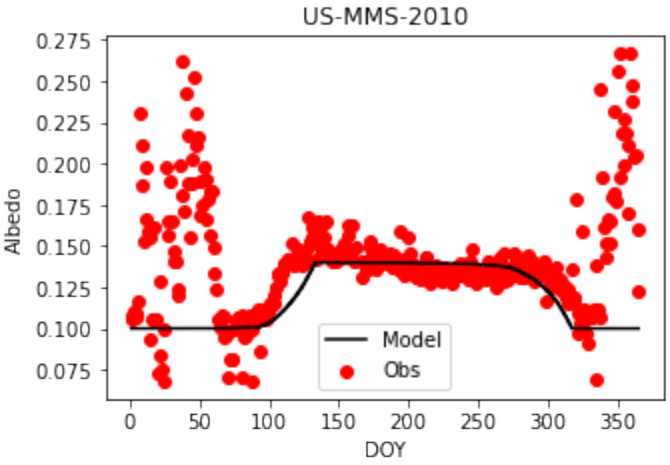
3.3.1 DecTr

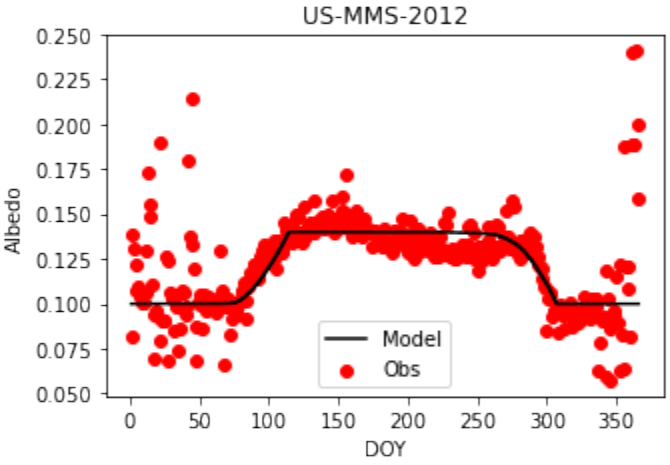
US-MMS

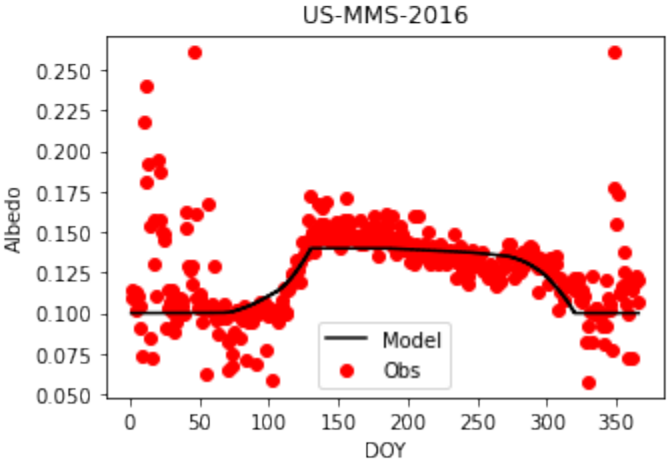
[28]:
```
name='US-MMS'
years=[2017]
read_plot(years,name)
```



[29]:
```
name='US-MMS'
years=[2010,2012,2016]
read_plot(years,name)
```


US-MMS-2010
Albedo
DOY
Model
Obs

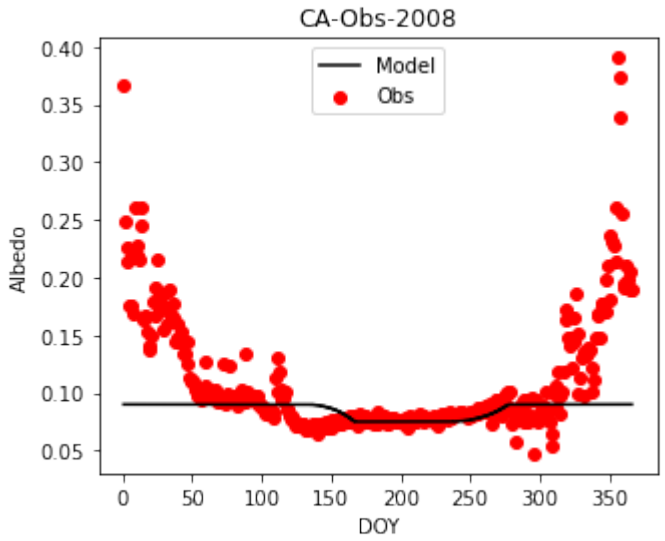

US-MMS-2012
Albedo
DOY
Model
Obs


US-MMS-2016
Albedo
DOY
Model
Obs

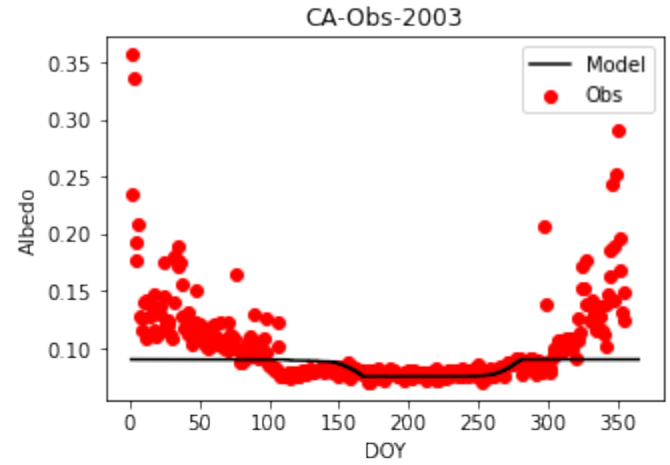
3.3.2 EveTr

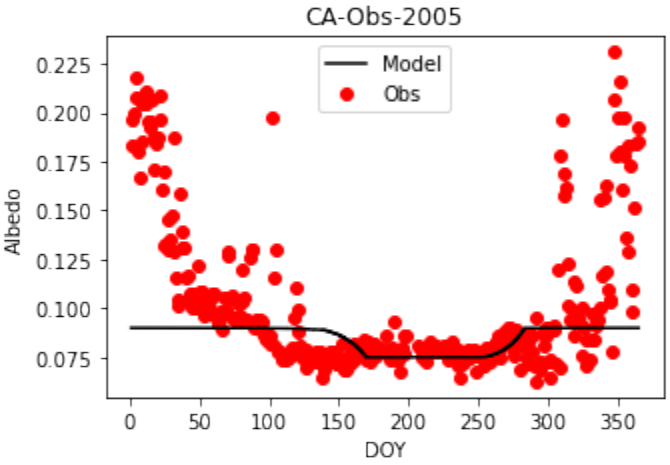
CA-Obs

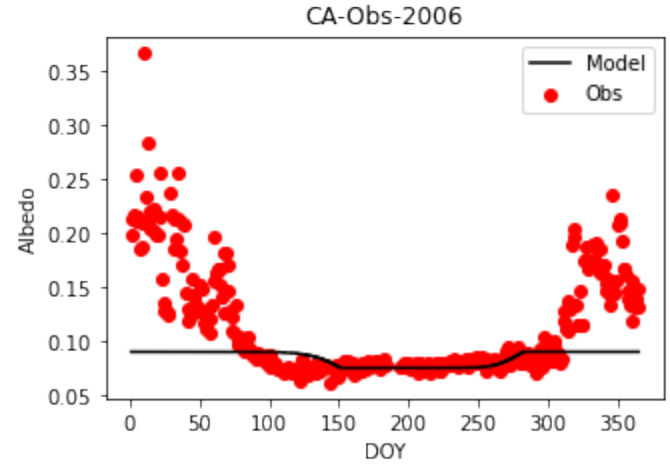
[30]:
```
name='CA-Obs'
years=[2008]
read_plot(years,name)
```



[31]:
```
name='CA-Obs'
years=[2003,2005,2006]
read_plot(years,name)
```



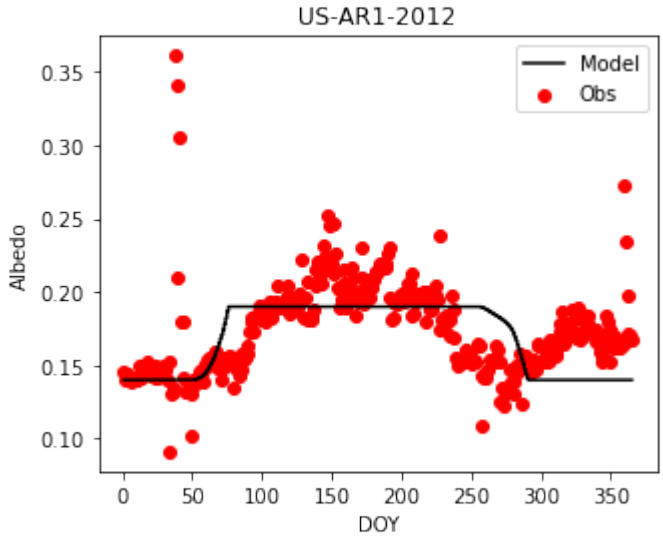




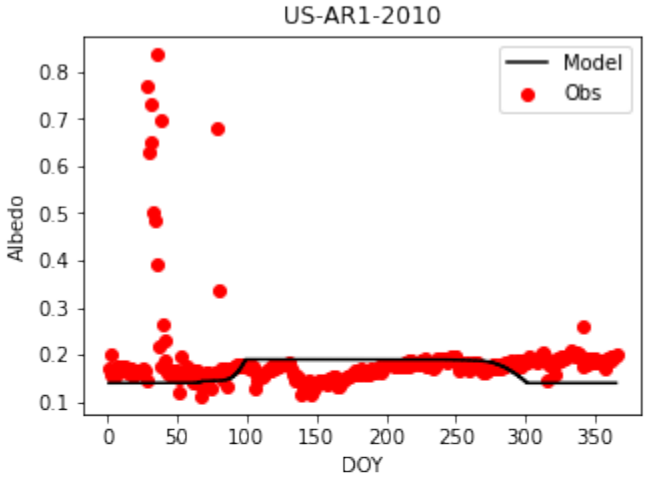
3.3.3 Grass

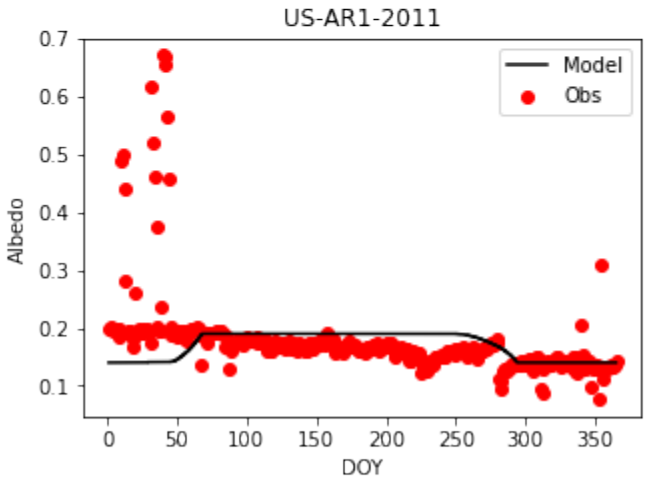
US-AR1

[32]:
```
name='US-AR1'
years=[2012]
read_plot(years,name)
```



[33]:
```
name='US-AR1'
years=[2010,2011]
read_plot(years,name)
```


US-AR1-2010
Model
Obs
Albedo
DOY
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
350


US-AR1-2011
Model
Obs
Albedo
DOY
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
350

CHAPTER 4

Roughness parameters

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pathlib import Path
from platypus.core import Problem
from platypus.types import Real,random
from platypus.algorithms import NSGAIII
import warnings
warnings.filterwarnings('ignore')
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[2]:
```
from z0_util import cal_vap_sat, cal_dens_dry, cal_dens_vap, cal_cpa, cal_dens_air,␣
↪cal_Lob
```

4.1 Function to calculate Neutral condition

[13]:
```
def cal_neutral(df_val,z_meas,h_sfc):

    # calculate Obukhov length
    ser_Lob = df_val.apply(
        lambda ser: cal_Lob(ser.H, ser.USTAR, ser.TA, ser.RH, ser.PA * 10), axis=1)

    # zero-plane displacement: estimated using rule f thumb `d=0.7*h_sfc`

    z_d = 0.7 * h_sfc

    if z_d >= z_meas:
        print(
            'vegetation height is greater than measuring height. Please fix this␣
↪before continuing'
```

(continues on next page)

(continued from previous page)

```
        )

    # calculate stability scale
    ser_zL = (z_meas - z_d) / ser_Lob

    # determine periods under quasi-neutral conditions
    limit_neutral = 0.01
    ind_neutral = ser_zL.between(-limit_neutral, limit_neutral)


    ind_neutral=ind_neutral[ind_neutral]
    df_sel = df_val.loc[ind_neutral.index, ['WS', 'USTAR']].dropna()
    ser_ustar = df_sel.USTAR
    ser_ws = df_sel.WS


    return ser_ws,ser_ustar
```

4.2 Function to calculate z0 and d using MO optimization

[14]:
```
def optimize_MO(df_val,z_meas,h_sfc):


    ser_ws,ser_ustar=cal_neutral(df_val,z_meas,h_sfc)

    def func_uz(params):
        z0=params[0]
        d=params[1]
        z = z_meas
        k = 0.4
        uz = (ser_ustar / k) * np.log((z - d) / z0)

        o1=abs(1-np.std(uz)/np.std(ser_ws))
        o2=np.mean(abs(uz-ser_ws))/(np.mean(ser_ws))

        return [o1,o2],[uz.min(),d-z0]

    problem = Problem(2,2,2)
    problem.types[0] = Real(0, 10)
    problem.types[1] = Real(0, h_sfc)


    problem.constraints[0] = ">=0"
    problem.constraints[1] = ">=0"

    problem.function = func_uz
    random.seed(12345)
    algorithm=NSGAIII(problem, divisions_outer=50)
    algorithm.run(30000)

    z0s=[]
    ds=[]
    os1=[]
    os2=[]
```

(continues on next page)

(continued from previous page)

```
    for s in algorithm.result:
        z0s.append(s.variables[0])
        ds.append(s.variables[1])
        os1.append(s.objectives[0])
        os2.append(s.objectives[1])

    idx=os2.index(min(os2, key=lambda x:abs(x-np.mean(os2))))
    z0=z0s[idx]
    d=ds[idx]

    return z0,d,ser_ws,ser_ustar
```

4.3 Loading data, cleaning and getting ready for optimization

[15]:
```
name_of_site='US-MMS'
years=[2010,2012,2016]


df_attr=pd.read_csv('all_attrs.csv')
z_meas=df_attr[df_attr.site==name_of_site].meas.values[0]
h_sfc=df_zmeas=df_attr[df_attr.site==name_of_site].height.values[0]
folder='data/data_csv_zip_clean_roughness/'
site_file = folder+'/'+name_of_site + '_clean.csv.gz'
df_data = pd.read_csv(site_file, index_col='time', parse_dates=['time'])
# Rain
bb=pd.DataFrame(~np.isin(df_data.index.date,df_data[df_data.P!=0].index.date))
bb.index=df_data.index
df_data=df_data[bb.values]


df_data=df_data[(df_data['WS']!=0)]

df_years=df_data.loc[f'{years[0]}']
for i in years[1:]:
    df_years=df_years.append(df_data.loc[f'{i}'])

df_val = df_years.loc[:, ['H', 'USTAR', 'TA', 'RH', 'PA', 'WS']].dropna()
df_val.head()
```

[15]:

time	H	USTAR	TA	RH	PA	WS
2010-01-01 00:00:00	21.873	0.739	-8.08	70.503	98.836	3.695
2010-01-01 01:00:00	41.819	0.855	-9.17	72.757	98.880	3.928
2010-01-01 02:00:00	-6.078	0.699	-9.63	72.611	98.910	3.088
2010-01-01 03:00:00	-16.788	0.581	-10.03	73.868	98.970	3.623
2010-01-01 04:00:00	5.006	0.562	-10.36	74.242	99.030	3.474

4.4 Calculating z0 and d

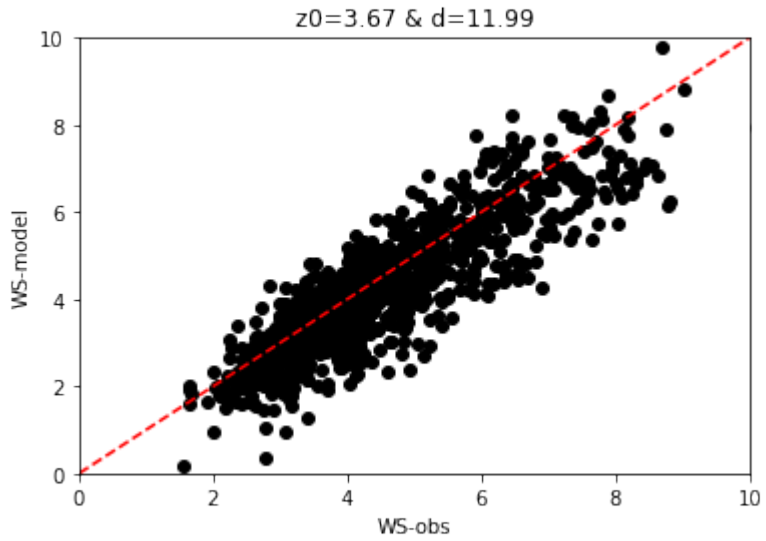
[16]:
```
z0,d,ser_ws,ser_ustar=optimize_MO(df_val,z_meas,h_sfc)
```

4.5 Calculating model wind speed using logarithmic law

[17]:
```
def uz(z0,d,ser_ustar,z_meas):
    z = z_meas
    k = 0.4
    uz = (ser_ustar / k) * np.log((z - d) / z0)
    return uz

ws_model=uz(z0,d,ser_ustar,z_meas)
```

[18]:
```
plt.scatter(ser_ws,ws_model,color='k')
plt.xlabel('WS-obs')
plt.ylabel('WS-model')
plt.title(f'z0={np.round(z0,2)} & d={np.round(d,2)}')
plt.plot([0,10],[0,10],color='r',linestyle='--')
plt.ylim([0,10])
plt.xlim([0,10])
```

[18]:
```
(0.0, 10.0)
```



CHAPTER 5

Roughness parameters (SuPy)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import supy as sp
import warnings
warnings.filterwarnings('ignore')
```

5.1 Loading data, cleaning and getting ready for optimization

[2]:
```
name_of_site='US-MMS'
years=[2010,2012,2016]


df_attr=pd.read_csv('all_attrs.csv')
z_meas=df_attr[df_attr.site==name_of_site].meas.values[0]
h_sfc=df_zmeas=df_attr[df_attr.site==name_of_site].height.values[0]
folder='data/data_csv_zip_clean_roughness/'
site_file = folder+'/'+name_of_site + '_clean.csv.gz'
df_data = pd.read_csv(site_file, index_col='time', parse_dates=['time'])
# Rain
bb=pd.DataFrame(~np.isin(df_data.index.date,df_data[df_data.P!=0].index.date))
bb.index=df_data.index
df_data=df_data[bb.values]


df_data=df_data[(df_data['WS']!=0)]

df_years=df_data.loc[f'{years[0]}']
for i in years[1:]:
    df_years=df_years.append(df_data.loc[f'{i}'])
```

(continues on next page)

(continued from previous page)

```
df_val = df_years.loc[:, ['H', 'USTAR', 'TA', 'RH', 'PA', 'WS']].dropna()
df_val.head()
```

[2]:

time	H	USTAR	TA	RH	PA	WS
2010-01-01 00:00:00	21.873	0.739	-8.08	70.503	98.836	3.695
2010-01-01 01:00:00	41.819	0.855	-9.17	72.757	98.880	3.928
2010-01-01 02:00:00	-6.078	0.699	-9.63	72.611	98.910	3.088
2010-01-01 03:00:00	-16.788	0.581	-10.03	73.868	98.970	3.623
2010-01-01 04:00:00	5.006	0.562	-10.36	74.242	99.030	3.474

5.2 Running supy to calculate z0 and d

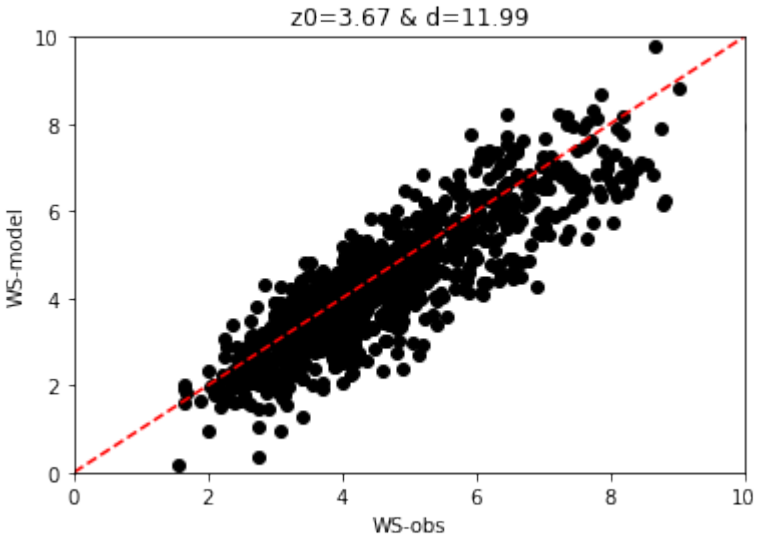
[3]:
```
z0,d,ser_ws,ser_ustar=sp.util.optimize_MO(df_val,z_meas,h_sfc)
```

5.3 Calculating model wind speed using logarithmic law

[4]:
```
def uz(z0,d,ser_ustar,z_meas):
    z = z_meas
    k = 0.4
    uz = (ser_ustar / k) * np.log((z - d) / z0)
    return uz

ws_model=uz(z0,d,ser_ustar,z_meas)
```

[5]:
```
plt.scatter(ser_ws,ws_model,color='k')
plt.xlabel('WS-obs')
plt.ylabel('WS-model')
plt.title(f'z0={np.round(z0,2)} & d={np.round(d,2)}')
plt.plot([0,10],[0,10],color='r',linestyle='--')
plt.ylim([0,10])
plt.xlim([0,10])
```

[5]:
```
(0.0, 10.0)
```


z0=3.67 & d=11.99
WS-model
WS-obs
0
2
4
6
8
10

CHAPTER 6

Surface conductance parameters

[7]:
```
from lmfit import Model, Parameters, Parameter
import numpy as np
import pandas as pd
from atmosp import calculate as ac
import numpy as np
from scipy.optimize import minimize
from pathlib import Path
import supy as sp
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from platypus.core import *
from platypus.types import *
from platypus.algorithms import *
import random
import pickle
import os
from shutil import copyfile
import warnings
warnings.filterwarnings('ignore')
```

This is a custom package specifically designed for this analysis. It contains various functions for reading and computing and plotting.

[4]:
```
from gs_util import read_forcing,modify_attr,cal_gs_obs,IQR_compare,obs_sim,cal_gs_
↪mod,gs_plot_test,modify_attr_2,func_parse_date
```

6.1 Preparing the data (obs and model)

[5]:
```
name='US-MMS'
year=2017
df_forcing= read_forcing(name,year)
```

[8]:
```
path_runcontrol = Path('runs/run'+'/') / 'RunControl.nml'
df_state_init = sp.init_supy(path_runcontrol)
df_state_init,level=modify_attr(df_state_init,name)
df_state_init.loc[:,'soilstore_id']=[50,50,50,50,50,50,0]
grid = df_state_init.index[0]
df_forcing_run = sp.load_forcing_grid(path_runcontrol, grid)
```

```
2020-03-26 10:09:25,798 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:09:26,642 -- SuPy -- INFO -- All cache cleared.
```

6.2 Spin up to get soil moisture

[9]:
```
error=10
for i in range(10):

    if (error <= 0.1):
        break
    df_output, df_state_final = sp.run_supy(df_forcing_run, df_state_init, save_
↪state=False)
    final_state = df_state_final[df_state_init.columns.levels[0]].iloc[1]
    df_state_init.iloc[0] = final_state
    soilstore_before = df_state_final.soilstore_id.iloc[0]
    soilstore_after = df_state_final.soilstore_id.iloc[1]
    diff_soil = sum(abs(soilstore_after-soilstore_before))
    error = 100*diff_soil/soilstore_before.mean()
    print(error)
```

```
2020-03-26 10:09:34,245 -- SuPy -- INFO -- ====================
2020-03-26 10:09:34,246 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:09:34,248 -- SuPy -- INFO --   Start: 2016-12-31 23:05:00
2020-03-26 10:09:34,250 -- SuPy -- INFO --   End: 2017-12-31 23:00:00
2020-03-26 10:09:34,251 -- SuPy -- INFO --
2020-03-26 10:09:34,253 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:09:34,254 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:09:48,656 -- SuPy -- INFO -- Execution time: 14.4 s
2020-03-26 10:09:48,656 -- SuPy -- INFO -- ====================

933.0266258519457
2020-03-26 10:09:49,106 -- SuPy -- INFO -- ====================
2020-03-26 10:09:49,107 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:09:49,107 -- SuPy -- INFO --   Start: 2016-12-31 23:05:00
2020-03-26 10:09:49,108 -- SuPy -- INFO --   End: 2017-12-31 23:00:00
2020-03-26 10:09:49,109 -- SuPy -- INFO --
2020-03-26 10:09:49,111 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:09:49,112 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:10:02,928 -- SuPy -- INFO -- Execution time: 13.8 s
2020-03-26 10:10:02,929 -- SuPy -- INFO -- ====================

0.0
```

6.3 Preparation of the data for model optimization

[10]:
```
df=df_output.SUEWS.loc[grid,:]
df=df.resample('1h',closed='left',label='right').mean()
```

[11]:
```
df_forcing.xsmd=df.SMD
df_forcing.lai=df.LAI
df_forcing = df_forcing[df_forcing.qe > 0]
df_forcing = df_forcing[df_forcing.qh > 0]
df_forcing = df_forcing[df_forcing.kdown > 5]
df_forcing = df_forcing[df_forcing.Tair > -20]
df_forcing.pres *= 1000
df_forcing.Tair += 273.15
gs_obs = cal_gs_obs(df_forcing.qh, df_forcing.qe, df_forcing.Tair,
                        df_forcing.RH, df_forcing.pres,df.RA)
df_forcing=df_forcing[gs_obs>0]
gs_obs=gs_obs[gs_obs>0]
df_forcing=df_forcing.replace(-999,np.nan)
```

[12]:
```
g_max=np.percentile(gs_obs,99)
s1=5.56
```

[13]:
```
print('Initial g_max is {}'.format(g_max))
```

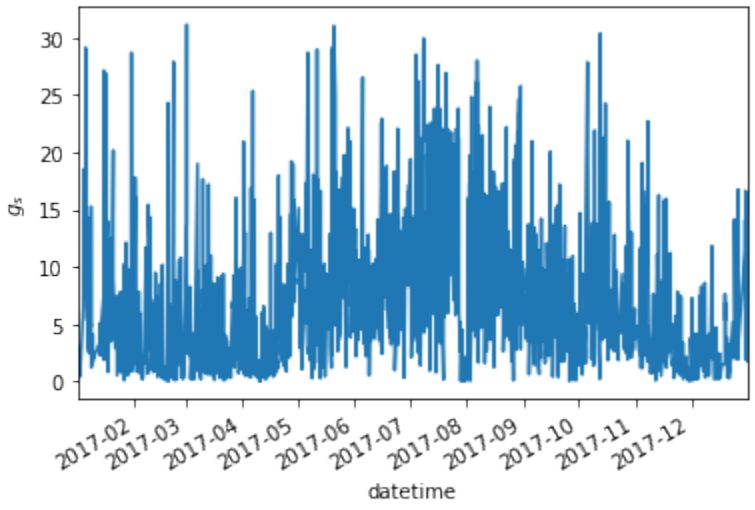
```
Initial g_max is 33.023283412991034
```

[14]:
```
df_forcing=df_forcing[gs_obs<g_max]
lai_max=df_state_init.laimax.loc[grid,:][1]
gs_obs=gs_obs[gs_obs<g_max]
```

6.3.1 Distribution of observed g_s

[16]:
```
gs_obs.plot()
plt.ylabel('$g_s$')
```

[16]:
```
Text(0, 0.5, '$g_s$')
```



[17]:
```
print('lai_max is {}'.format(lai_max))
```

```
lai_max is 5.0
```

6.4 Splitting the data to test and train

[19]:
```
df_forcing_train, df_forcing_test, gs_train, gs_test = train_test_split(df_forcing,␣
↪gs_obs, test_size=0.6, random_state=42)
```

[20]:
```
kd=df_forcing_train.kdown
ta=df_forcing_train.Tair
rh=df_forcing_train.RH
pa=df_forcing_train.pres
smd=df_forcing_train.xsmd
lai=df_forcing_train.lai
```

6.5 Optimization

More info in here

6.5.1 Function to optimize

[22]:
```
def fun_to_opts(G):
    [g1,g2, g3, g4, g5, g6]=[G[0],G[1],G[2],G[3],G[4],G[5]]
    gs_model,g_lai,g_kd,g_dq,g_ta,g_smd,g_z=cal_gs_mod(kd, ta, rh, pa, smd, lai,
                          [g1, g2, g3, g4, g5, g6],
                          g_max, lai_max, s1)
    gs_obs=gs_train
    o1=abs(1-np.std(gs_model)/np.std(gs_obs)) # normilized std difference
    o2=np.mean(abs(gs_model-gs_obs))/(np.mean(gs_obs))
    return [o1,o2],[gs_model.min()]
```

6.5.2 Problem definition and run

[27]:
```
problem = Problem(6,2,1)
problem.types[0] = Real(.09, .5)
problem.types[1] = Real(100, 500)
problem.types[2] = Real(0, 1)
problem.types[3] = Real(0.4, 1)
problem.types[4] = Real(25, 55)
problem.types[5] = Real(0.02, 0.03)

problem.constraints[0] = ">=0"

problem.function = fun_to_opts
random.seed(12345)
algorithm=CMAES(problem, epsilons=[0.005])
algorithm.run(3000)
```

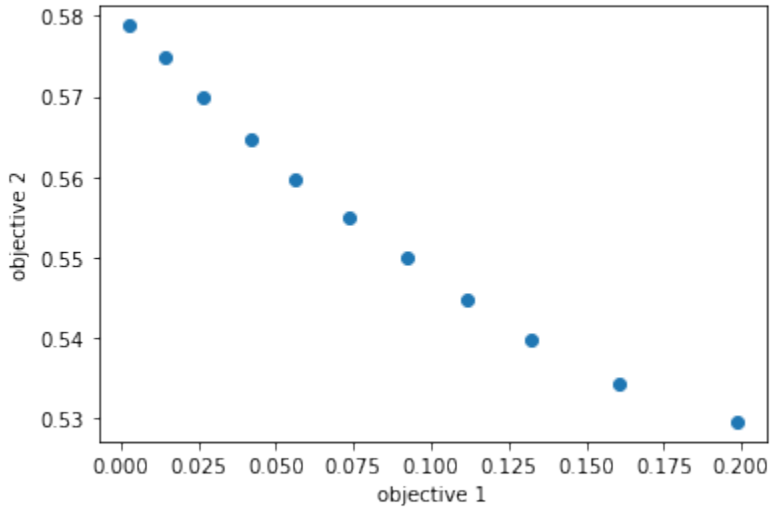
6.5.3 Solutions

[28]:
```
print( " Obj1\t Obj2")

for solution in algorithm.result[:10]:
    print ("%0.3f\t%0.3f" % tuple(solution.objectives))
```

```
 Obj1    Obj2
0.073   0.555
0.112   0.545
0.161   0.534
0.056   0.560
0.003   0.579
0.092   0.550
0.199   0.530
0.133   0.540
0.026   0.570
0.014   0.575
```

[29]:
```
f, ax = plt.subplots(1, 1)
plt.scatter([s.objectives[0] for s in algorithm.result],
            [s.objectives[1] for s in algorithm.result])
plt.xlabel('objective 1')
plt.ylabel('objective 2')
```

[29]:
```
Text(0, 0.5, 'objective 2')
```



[30]:
```
all_std=[s.objectives[0] for s in algorithm.result]
all_MAE=[s.objectives[1] for s in algorithm.result]
all_std=np.array(all_std)
all_MAE=np.array(all_MAE)
```

6.5.4 Choosing between : the median of two objectives, where obj1 is max or where obj2 is max

[31]:
```
method='median' # 'obj1' or 'obj2' or 'median'
colors= ['b','g','r','y']
```

(continues on next page)

(continued from previous page)

```
if method == 'median':
    idx_med=np.where(all_MAE==all_MAE[(all_std<=np.median(all_std))].min())[0][0]
elif method == 'obj1':
    idx_med=np.where(all_MAE==all_MAE[(all_std>=np.min(all_std))].max())[0][0]
elif method == 'obj2':
    idx_med=np.where(all_MAE==all_MAE[(all_std<=np.max(all_std))].min())[0][0]
print(all_std[idx_med])
print(all_MAE[idx_med])
```

```
0.07329333444496633
0.5549996145597578
```

[32]:
```
[g1,g2,g3,g4,g5,g6] = algorithm.result[idx_med].variables
```

6.5.5 Saving the solution

[33]:
```
with open('outputs/g1-g6/'+name+'-g1-g6.pkl','wb') as f:
    pickle.dump([g1,g2,g3,g4,g5,g6], f)
```

[34]:
```
with open('outputs/g1-g6/'+name+'-g1-g6.pkl','rb') as f:
    [g1,g2,g3,g4,g5,g6]=pickle.load(f)
```

[35]:
```
pd.DataFrame([np.round([g1,g2,g3,g4,g5,g6],3)],columns=['g1','g2','g3','g4','g5','g6
↪'],index=[name])
```

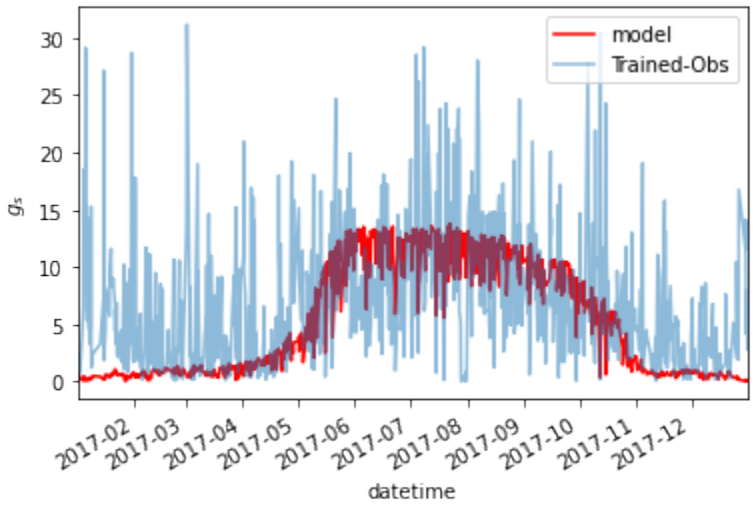
[35]:

	g1	g2	g3	g4	g5	g6
US-MMS	0.431	104.34	0.634	0.683	35.25	0.03

Let's see how the model and observation compare for the training data set:

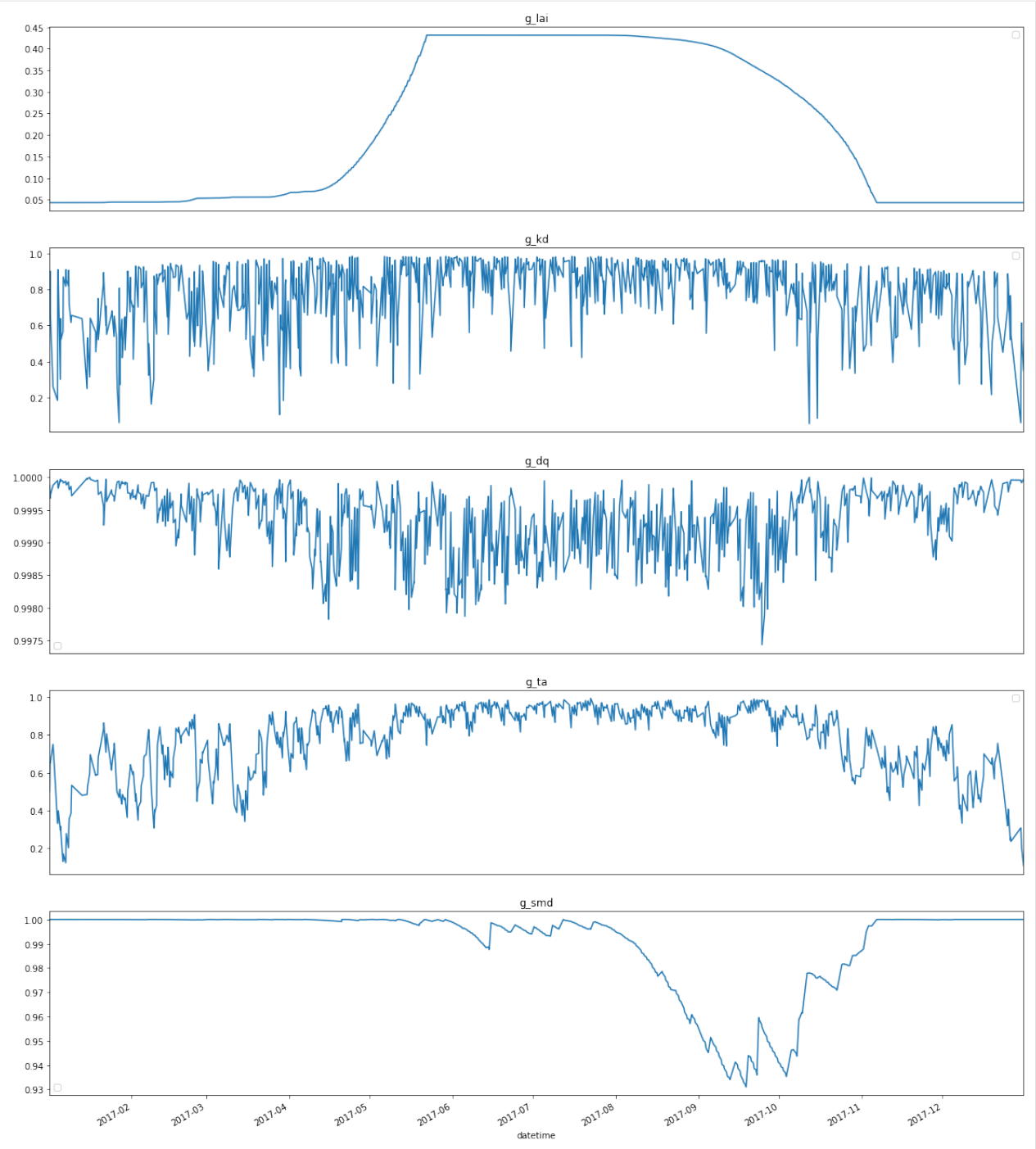
[38]:
```
gs_model,g_lai,g_kd,g_dq,g_ta,g_smd,g_max=cal_gs_mod(kd, ta, rh, pa, smd, lai,
                          [g1, g2, g3, g4, g5, g6],
                          g_max, lai_max, s1)

gs_model.plot(color='r',label='model')
gs_train.plot(alpha=0.5,label='Trained-Obs')
plt.legend()
plt.ylabel('$g_s$')
```

[38]:
```
Text(0, 0.5, '$g_s$')
```



Let's look at each individual g term:

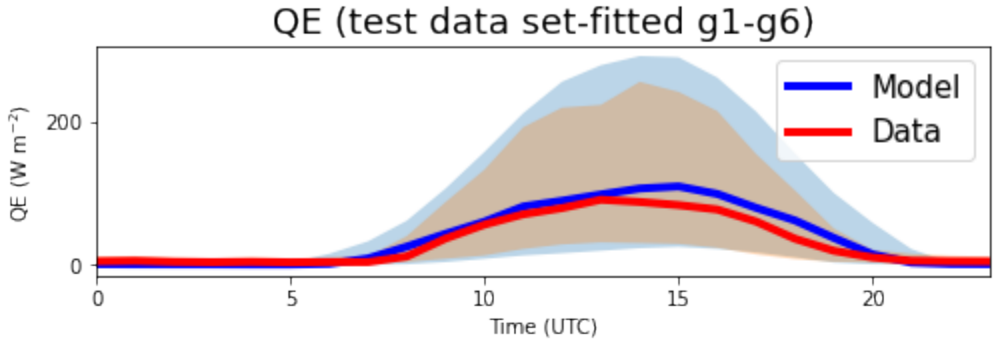
[39]:
```
g_dq=pd.DataFrame(g_dq,index=g_lai.index)
fig,axs=plt.subplots(5,1,figsize=(20,25))
a={'0':g_lai,'1':g_kd,'2':g_dq,'3':g_ta,'4':g_smd}
b={'0':'g_lai','1':'g_kd','2':'g_dq','3':'g_ta','4':'g_smd'}
for i in range(0,5):
    ax=axs[i]
    a[str(i)].plot(ax=ax)
    ax.set_title(b[str(i)])
    ax.legend('')
    if i!=4:
        ax.set_xticks([''])
        ax.set_xlabel('')
```

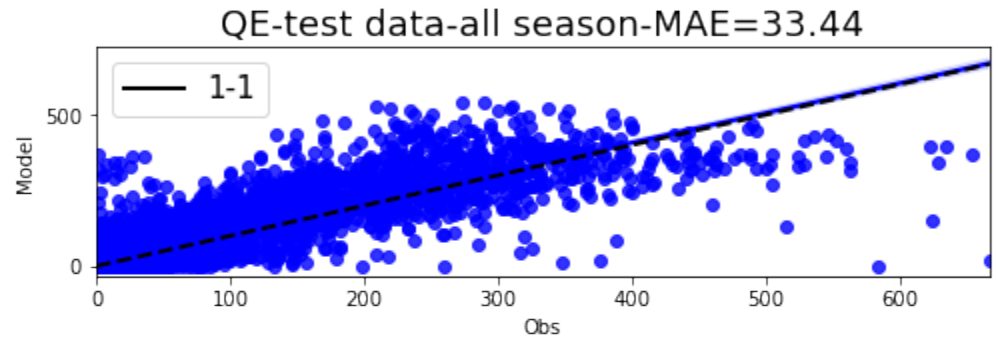


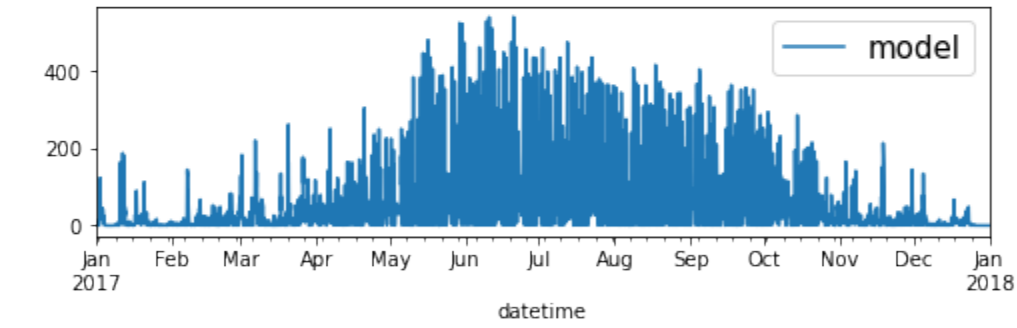
6.6 Running Supy with new g1-g6

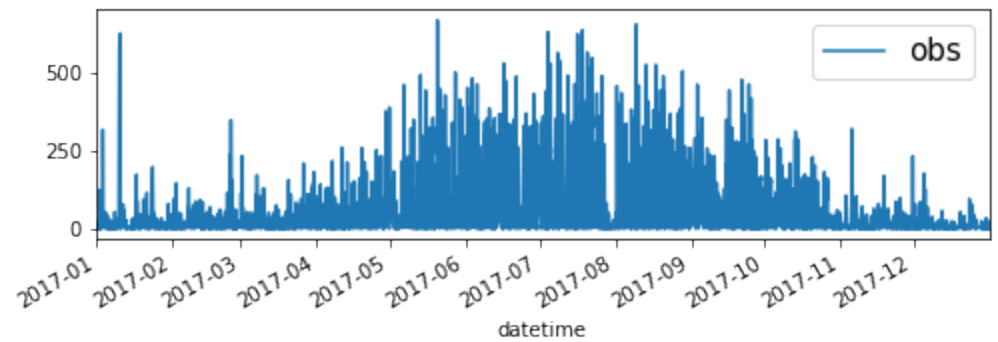
[40]:
```
alpha=2.6 # need to be tuned iteratively
name='US-MMS'
year=year
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year,alpha)
```

```
2020-03-26 10:38:44,539 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:38:45,412 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:38:47,776 -- SuPy -- INFO -- ====================
2020-03-26 10:38:47,776 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:38:47,777 -- SuPy -- INFO --   Start: 2016-12-31 23:05:00
2020-03-26 10:38:47,778 -- SuPy -- INFO --   End: 2017-12-31 23:00:00
2020-03-26 10:38:47,779 -- SuPy -- INFO --
2020-03-26 10:38:47,780 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:38:47,782 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:39:08,180 -- SuPy -- INFO -- Execution time: 20.4 s
2020-03-26 10:39:08,181 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
Model
Data
QE (W m^{-2})
200
0
0
5
10
15
20
Time (UTC)


QE-test data-all season-MAE=33.44
1-1
Model
500
0
0
100
200
300
400
500
600
Obs


model
400
200
0
Jan
2017
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
2018
datetime


obs
500
250
0
2017-01
2017-02
2017-03
2017-04
2017-05
2017-06
2017-07
2017-08
2017-09
2017-10
2017-11
2017-12
datetime

[41]:
```
g1=g1*alpha
g_max=g1*g_max
g1=1
```

[42]:
```
g_max
```

[42]:
```
37.03471400427182
```

6.7 Creating the table for all sites here (if more than one site is tuned)

[43]:
```
sites=['US-MMS']
g1_g6_all=pd.DataFrame(columns=['g1','g2','g3','g4','g5','g6'])

for s in sites:
    with open('outputs/g1-g6/'+s+'-g1-g6.pkl','rb') as f:
        g1_g6_all.loc[s,:]=pickle.load(f)
g1_g6_all
```

[43]:

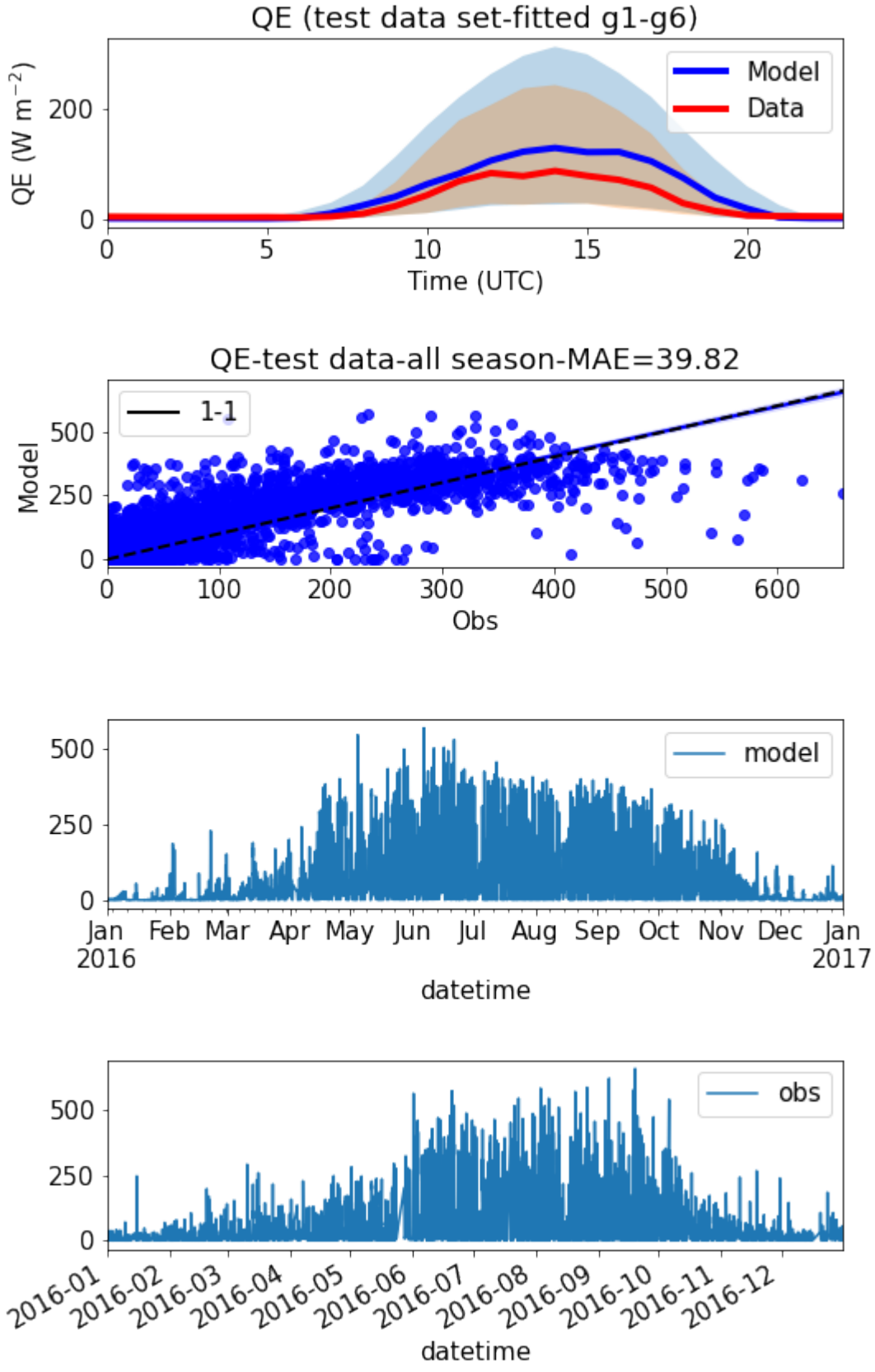
	g1	g2	g3	g4	g5	g6
US-MMS	0.431336	104.34	0.633861	0.682953	35.25	0.0298504

6.8 To test

6.8.1 US-MMS-2016

[44]:
```
g1,g2,g3,g4,g5,g6=g1_g6_all.loc['US-MMS',:].values
g_max=g_max
g1=1
s1=5.56


name='US-MMS'
year=2016
df_forcing= read_forcing(name,year)
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year)
```

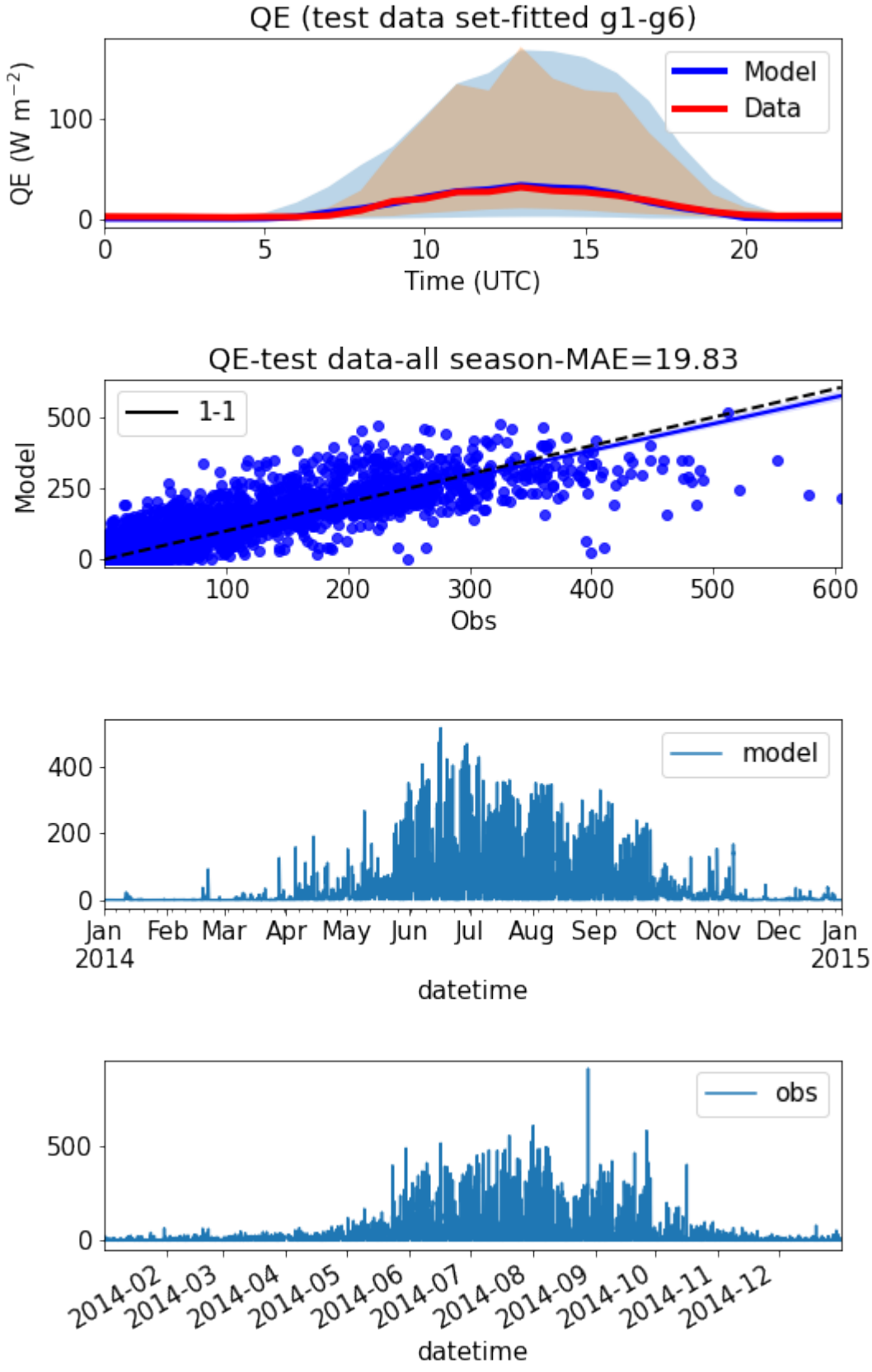
```
2020-03-26 10:41:58,297 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:41:59,122 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:42:01,787 -- SuPy -- INFO -- ====================
2020-03-26 10:42:01,789 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:42:01,791 -- SuPy -- INFO --   Start: 2015-12-31 23:05:00
2020-03-26 10:42:01,801 -- SuPy -- INFO --   End: 2016-12-31 23:00:00
2020-03-26 10:42:01,802 -- SuPy -- INFO --
2020-03-26 10:42:01,807 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:42:01,810 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:42:24,960 -- SuPy -- INFO -- Execution time: 23.2 s
2020-03-26 10:42:24,961 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
QE (W m^{-2})
Model
Data
Time (UTC)
QE-test data-all season-MAE=39.82
1-1
Model
Obs
model
datetime
obs
datetime

6.8.2 UMB-2014

[46]:
```
g1,g2,g3,g4,g5,g6=g1_g6_all.loc['US-MMS',:].values
g_max=g_max
g1=1
s1=5.56

name='US-UMB'
year=2014
df_forcing= read_forcing(name,year)
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year)
```

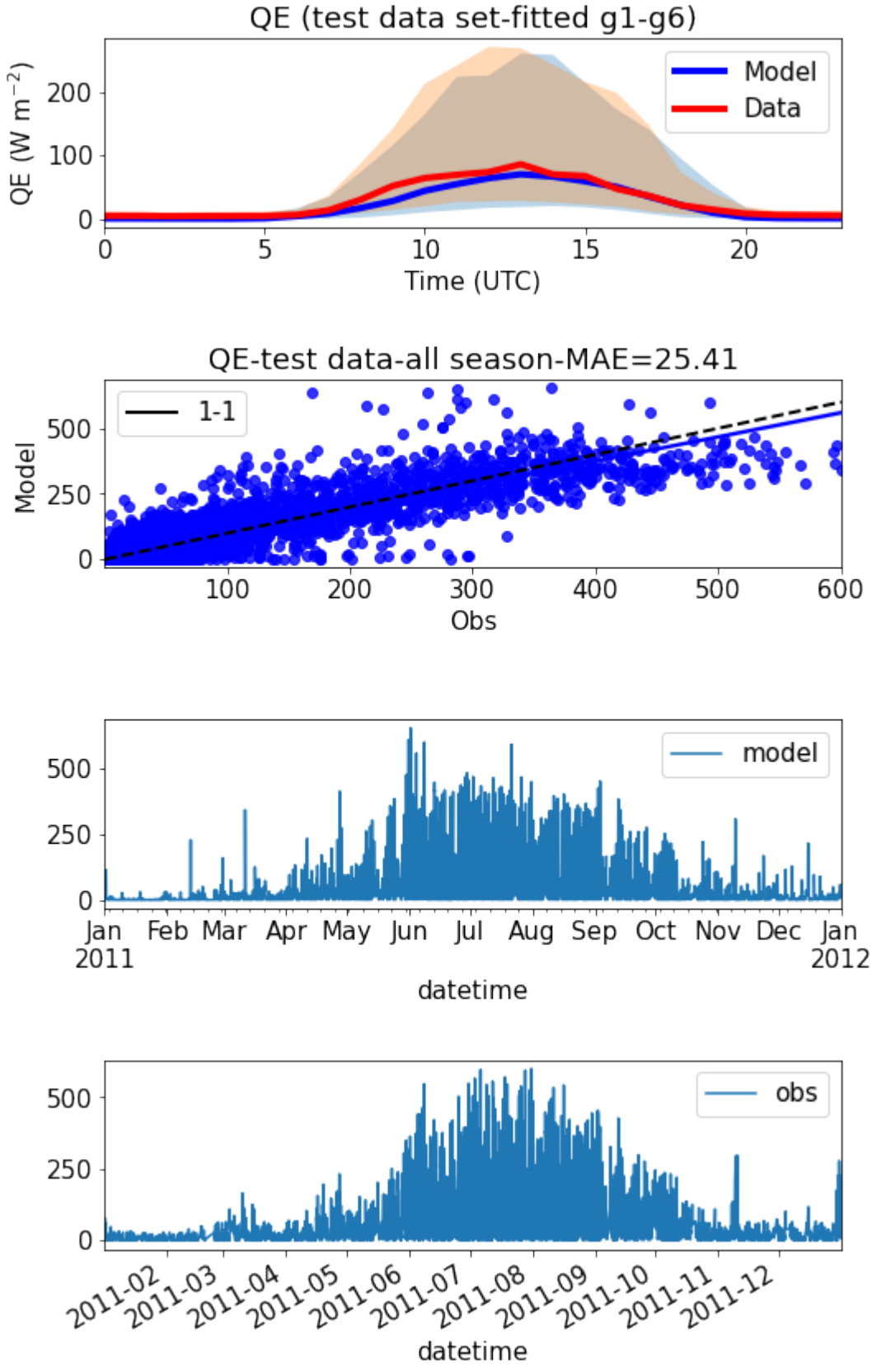
```
2020-03-26 10:43:28,350 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:29,145 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:31,505 -- SuPy -- INFO -- ====================
2020-03-26 10:43:31,506 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:43:31,506 -- SuPy -- INFO --   Start: 2013-12-31 23:05:00
2020-03-26 10:43:31,507 -- SuPy -- INFO --   End: 2014-12-31 23:00:00
2020-03-26 10:43:31,509 -- SuPy -- INFO --
2020-03-26 10:43:31,511 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:43:31,512 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:43:45,433 -- SuPy -- INFO -- Execution time: 13.9 s
2020-03-26 10:43:45,434 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
QE (W m^{-2})
Model
Data
Time (UTC)
QE-test data-all season-MAE=19.83
1-1
Model
Obs
model
datetime
obs
datetime

6.8.3 US-Oho-2011

[47]:
```
g1,g2,g3,g4,g5,g6=g1_g6_all.loc['US-MMS',:].values
g_max=g_max
g1=1
s1=5.56

name='US-Oho'
year=2011
df_forcing= read_forcing(name,year)
gs_plot_test(g1,g2,g3,g4,g5,g6,g_max,s1,name,year)
```

```
2020-03-26 10:43:49,647 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:50,493 -- SuPy -- INFO -- All cache cleared.
2020-03-26 10:43:53,625 -- SuPy -- INFO -- ====================
2020-03-26 10:43:53,626 -- SuPy -- INFO -- Simulation period:
2020-03-26 10:43:53,627 -- SuPy -- INFO --   Start: 2010-12-31 23:05:00
2020-03-26 10:43:53,628 -- SuPy -- INFO --   End: 2011-12-31 23:00:00
2020-03-26 10:43:53,629 -- SuPy -- INFO --
2020-03-26 10:43:53,631 -- SuPy -- INFO -- No. of grids: 1
2020-03-26 10:43:53,632 -- SuPy -- INFO -- SuPy is running in serial mode
2020-03-26 10:44:11,821 -- SuPy -- INFO -- Execution time: 18.2 s
2020-03-26 10:44:11,822 -- SuPy -- INFO -- ====================
```


QE (test data set-fitted g1-g6)
QE (W m^{-2})
Model
Data
Time (UTC)
QE-test data-all season-MAE=25.41
1-1
Model
Obs
model
datetime
obs
2011-02
2011-03
2011-04
2011-05
2011-06
2011-07
2011-08
2011-09
2011-10
2011-11
2011-12
datetime