

SAM Analysis

Designed to calculate and plot the SAM. Link to [Data analysis](#).

Choices for data

In [1]:

```
#Experiment data for analysis
dataVariableId = 'psl'
dataSourceId = "UKESM1-0-LL"
experimentId = "historical"
tableId = "Amon"
approvedIds = ["r1i1p1f2", "r2i1p1f2", "r3i1p1f2"] #insert start of approved member_ids

#External data files
externalSAMFileName = "SAMIndex.txt"
GMMModelNames = ["GMM_UK_2Class_R1", "GMM_UK_2Class_R2", "GMM_UK_2Class_R3"], "GMM_UK_2Clas

#Custom Variables
latSel = slice(-89.5,-29.5) #Selected latitude to be investigated
maskName = "OceanMaskVolcello"
```

Imports

In [2]:

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import dask.dataframe as dd
import fsspec
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xarray as xr
import zarr

from dask import delayed
from matplotlib.pyplot import cm
from scipy import signal
print("Imports complete")
```

Imports complete

Data loading

Loading the ESM Pressure data

In [3]:

```
df = pd.read_csv('https://storage.googleapis.com/cmip6/cmip6-zarr-consolidated-stores.csv')
dfFilt = df[df.variable_id.eq(dataVariableId) & df.source_id.eq(dataSourceId) & df.table_id.eq(tableId)

memberArr = np.empty(shape=0, dtype=bool)
for i in dfFilt["member_id"]:
    rowSel = i[:] in approvedIds #adapt i[:] to match size of approvedIds
    memberArr = np.append(memberArr, rowSel)

memberSer = pd.Series(memberArr, name='bools')
dfFilt = dfFilt[memberSer.values]
```

```

fileSetList = []
for i in range(len(dfFilt)):
    zstore = dfFilt.zstore.values[i]
    mapper = fsspec.get_mapper(zstore)
    fileRaw = xr.open_zarr(mapper, consolidated=True)
    fileSetList.append(fileRaw)
fileCount = len(fileSetList)
if fileCount:
    print(str(fileCount)+" "+dataSourceId+" data sets opened")
else:
    print("No ESM3 data sets opened")

for i in range(fileCount): #Formatting dates into np.datetime64 format
    startDateIterate = np.datetime64(fileSetList[i]['time'].values[0], 'M')
    endDateIterate = np.datetime64(fileSetList[i]['time'].values[-1], 'M') + np.timedelta64(1, 'M')
    fileSetList[i]['time']=('time', np.arange(startDateIterate, endDateIterate, dtype='datetime64[M]'))
    fileSetList[i]['time_bnds']=('time_bnds', np.arange(startDateIterate, endDateIterate, np.timedelta64(1, 'M')))
fileSet = xr.combine_nested(fileSetList, concat_dim='RunId') #Combining data sets
print("Data sets successfully merged")

dfESMLat = fileSet.psl.sel(lat=latSel) #Selection of latitude
dfESMLat = dfESMLat.reset_coords(drop=True) #Removes lev if single value

globalStartDate = dfESMLat["time"] [0].values
globalDateInc = dfESMLat["time"] [1].values - globalStartDate
globalEndDateIn = dfESMLat["time"] [-1].values
globalEndDateOut = globalEndDateIn + globalDateInc

globalStartDateStr = str(globalStartDate) [:7]
globalEndDateInStr = str(globalEndDateIn) [:7]
globalEndDateOutStr = str(globalEndDateOut) [:7]

print("UKESM data sucessfully loaded.")

```

3 UKESM1-0-LL data sets opened
Data sets successfully merged
UKESM data sucessfully loaded.

Loading the ocean mask

In [4]:

```

maskFile = xr.open_dataset(maskName)
oceanMask = maskFile.to_array()
print("Mask Loaded into oceanMask")

```

Mask Loaded into oceanMask

Unpacking ocean masks

In [5]:

```

#Mask unpacking cell
geoRange = oceanMask #copying mask
geoRange = geoRange.rename({"variable":"cleanMe"}) #Dimension removal
geoRange = geoRange.sel(cleanMe = geoRange.cleanMe.values[0]) #Dimension removal
geoRange = geoRange.reset_coords("cleanMe", drop=True) #Dimension removal
geoRangeS = geoRange.stack(ij =("i", "j")) #Stacking
geoRangeFilt = geoRangeS.dropna("ij")
print("Ocean mask unpacked into geoRangeFilt.")

```

Ocean mask unpacked into geoRangeFilt.

Loading external SAM Index

```
In [6]:  
externalSAMIndex = np.loadtxt("SAMIndex.txt", skiprows=1, max_rows=64)  
externalSAMIndexYear = np.transpose(externalSAMIndex) [0]  
externalSAMIndexSort = np.empty(shape=(0,2))  
for i in range(len(externalSAMIndexYear)):  
    for j in range(1,13):  
        externalSAMIndexSort = np.append(externalSAMIndexSort, [[np.datetime64(str(int(ext  
externalSAMIndexSort = np.transpose(externalSAMIndexSort)  
print("External SAM index data in "+externalSAMFileName+" loaded into externalSAMIndexSort")
```

External SAM index data in SAMIndex.txt loaded into externalSAMIndexSort.

Loading mean latitudes from GMM

```
In [7]:  
thresholds = [0.25, 0.5, 0.75, 0.85]  
LatMeanList = []  
for i in GMModelNames:  
    importName = i + "_LatMeans.npy"  
    LatMeanList.append(np.load(importName))  
print("GMM mean latitudes imported into LatMeanList.")
```

GMM mean latitudes imported into LatMeanList.

SAM pressure calculations

```
In [8]:  
if True:  
    normalisationData = dfESMLat.sel(time=slice('1971-01','2000-12'))  
    normalisationDataP40 = normalisationData.sel(lat=-40, method="nearest")  
    selLat40 = normalisationDataP40["lat"].values  
    normalisationDataP65 = normalisationData.sel(lat=-65, method="nearest")  
    selLat65 = normalisationDataP65["lat"].values  
  
    normalisationDataP40MLon = normalisationDataP40.mean(["lon"], keep_attrs=True)  
    normalisationDataP65MLon = normalisationDataP65.mean(["lon"], keep_attrs=True)  
    normalisationDataP40MLonG = normalisationDataP40MLon.groupby("time.month")  
    normalisationDataP65MLonG = normalisationDataP65MLon.groupby("time.month")  
    P40m = normalisationDataP40MLonG.mean(["time"], keep_attrs=True).values  
    P65m = normalisationDataP65MLonG.mean(["time"], keep_attrs=True).values  
  
    normalisationDataP40G = normalisationDataP40.groupby("time.month")  
    P40std = normalisationDataP40MLonG.std(["time"], keep_attrs=True).values #  
    normalisationDataP65G = normalisationDataP65.groupby("time.month")  
    P65std = normalisationDataP65MLonG.std(["time"], keep_attrs=True).values #  
else:  
    normalisationData = dfESMLat.sel(time=slice('1971-01','2000-12'))  
    normalisationDataP40 = normalisationData.sel(lat=slice(-41,-39))  
    normalisationDataP40M = normalisationDataP40.mean("lat", keep_attrs=True)  
    normalisationDataP65 = normalisationData.sel(lat=slice(-66,-64))  
    normalisationDataP65M = normalisationDataP65.mean("lat", keep_attrs=True)  
  
    normalisationDataP40MLon = normalisationDataP40M.mean(["lon"], keep_attrs=True)  
    normalisationDataP65MLon = normalisationDataP65M.mean(["lon"], keep_attrs=True)  
    normalisationDataP40MLonG = normalisationDataP40MLon.groupby("time.month")  
    normalisationDataP65MLonG = normalisationDataP65MLon.groupby("time.month")  
    P40m = normalisationDataP40MLonG.mean(["time"], keep_attrs=True).values  
    P65m = normalisationDataP65MLonG.mean(["time"], keep_attrs=True).values  
  
    normalisationDataP40G = normalisationDataP40.groupby("time.month")  
    P40std = normalisationDataP40G.std(["lat","lon","time"], keep_attrs=True).values  
    normalisationDataP65G = normalisationDataP65.groupby("time.month")  
    P65std = normalisationDataP65G.std(["lat","lon","time"], keep_attrs=True).values  
print("Normalisation data calculated.")
```

Calculation Functions

Functions:

- externalSAMSearch - returns externalSAMIndexSort between two dates.
- SAMCalculateRaw - returns a data array of the difference in pressures between lat -40 and lat -65 (tolerance $\pm 1^\circ$) for an input data array, not normalised .
- SAMCalculateNorm - returns a data array of the difference in pressures between lat -40 and lat -65 (tolerance $\pm 1^\circ$) for an input data array, normalised using second input data array.

In [9]:

```

def butter_lowpass(data, cut, order=4, sample_freq=1) :
    nyq = 0.5*sample_freq
    pass_freq = 1./cut/nyq
    sos=signal.butter(order, pass_freq, 'low', output='sos')
    filt=signal.sosfiltfilt(sos,data)
    return filt

def externalSAMSearch(startDate, endDate):
    '''Performs np searching across externalSAMIndexSort for start-end date data, returns
    x, startIndex = np.where(externalSAMIndexSort == np.datetime64(startDate, 'M'))
    x, endIndex = np.where(externalSAMIndexSort == np.datetime64(endDate, 'M'))
    dateArray = externalSAMIndexSort[0][startIndex[0] : endIndex[0] + 1]
    dataArray = externalSAMIndexSort[1][startIndex[0] : endIndex[0] + 1]
    externalSAMSearch = np.append([dateArray], [dataArray], axis=0)
    return externalSAMSearch

def SAMCalculateNorm(dataArray, runId):
    '''Calculates the SAM index using P40m, P65m, P40std and P65std. Returns SAM index np
    timeArray = dataArray["time"].values
    startTimeY = timeArray[0].astype('datetime64[Y]').astype(int) + 1970
    endTimeY = timeArray[-1].astype('datetime64[Y]').astype(int) + 1970
    startTimeM = timeArray[0].astype('datetime64[M]').astype(int) % 12 + 1
    endTimeM = timeArray[-1].astype('datetime64[M]').astype(int) % 12 + 1

    P40 = dataArray.sel(lat=slice(-41, -39))
    P65 = dataArray.sel(lat=slice(-66, -64))

    P40zArray = np.empty(shape=0)
    P65zArray = np.empty(shape=0)
    if startTimeM > 1: #Part start year z transform
        P40YearPartsS = P40.sel(time = str(startTimeY))
        P40YearPartsS = P40YearPartsS.mean(dim=["lat", "lon"])
        P40YearPartsS = P40YearPartsS.reset_coords(drop=True)
        P40zYearPartsS = ((P40YearPartsS-P40m[startTimeM-1:]) / P40std[startTimeM-1:]).values
        P40zArray = np.append(P40zArray, P40zYearPartsS)

        P65YearPartsS = P65.sel(time = str(startTimeY))
        P65YearPartsS = P65YearPartsS.mean(dim=["lat", "lon"])
        P65YearPartsS = P65YearPartsS.reset_coords(drop=True)
        P65zYearPartsS = ((P65YearPartsS-P65m[startTimeM-1:]) / P65std[startTimeM-1:]).values
        P65zArray = np.append(P65zArray, P65zYearPartsS)
        startTimeY += 1

    P40zYearPartE = np.empty(0)
    P65zYearPartE = np.empty(0)
    if endTimeM < 12: #Part end year z transform
        P40YearPartE = P40.sel(time = str(endTimeY))

```

```

P40YearPartE = P40YearPartE.mean(dim=["lat", "lon"])
P40YearPartE = P40YearPartE.reset_coords(drop=True)
P40zYearPartE = ((P40YearPartE-P40m[:endTimeM-12])/P40std[:endTimeM-12]).values
endTimeY -= 1

if startTimeY <= endTimeY: #Full year z transform
    for i in range(startTimeY, endTimeY + 1):
        P40Year = P40.sel(time = str(i))
        P40Year = P40Year.mean(dim=["lat", "lon"])
        P40Year = P40Year.reset_coords(drop=True)
        P40zYear = ((P40Year-P40m[runId])/P40std[runId]).values
        P40zArray = np.append(P40zArray, P40zYear)

        P65Year = P65.sel(time = str(i))
        P65Year = P65Year.mean(dim=["lat", "lon"])
        P65Year = P65Year.reset_coords(drop=True)
        P65zYear = ((P65Year-P65m[runId])/P65std[runId]).values
        P65zArray = np.append(P65zArray, P65zYear)

P40zArray = np.append(P40zArray, P40zYearPartE) #Appends End year
P65zArray = np.append(P65zArray, P65zYearPartE) #Appends End year

SAMArray = P40zArray - P65zArray
SAMArray = np.round(SAMArray, 2)

try:
    intRunId = runId.values
except:
    intRunId = runId

indexName = "SAM_Index_"+str(intRunId)
exportSAMData = {"time":timeArray, indexName:SAMArray}
exportSAMDF = pd.DataFrame(exportSAMData, columns=["time", indexName])
return exportSAMDF

print("Calculation functions defined")

```

Calculation functions defined

Plotting Functions

Functions:

- mapPlot - plots the Southern Ocean orthographic project of the input data set, with input title and figure number.
- timePlot - plots the SAM index in the input data array over time.
- timePlotStd - plots the SAM index for a 12 month period (starting January) with error bars.
- timePlotYear - plots the SAM index for a 12 month period (starting January) without error bars. Compatible with timePlotStd.
- timePlotDecade - plots the SAM index for an input data array over a decade period. Start date is displayed as 2001-01.

In [10]:

```

months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"]
lineType = [".", "--", ":"]
timeAxis = np.arange(np.datetime64("1980-01"), np.datetime64("2010-01"), np.timedelta64(1,

```

```

def latMeanPlot (dataArray, labelIn, plotNo):
    plt.figure(plotNo, figsize=(20,10))
    plt.plot(timeAxis, dataArray, label=labelIn)
    plt.xlabel("Month")
    plt.ylabel("Mean Latitude")
    plt.legend()

def locationPlotXr (dataArray, size, plotNo):
    '''Plots locations of numpy arrays with date colour scheme'''
    plt.figure(plotNo, figsize=size)
    ax = plt.axes(projection=ccrs.SouthPolarStereo())
    ax.add_feature(cfeature.OCEAN)
    ax.add_feature(cfeature.COASTLINE)
    ax.coastlines()
    ax.gridlines()
    im = ax.scatter(dataArray["lon"], dataArray["lat"], transform=ccrs.PlateCarree())
    plt.plot(np.arange(0,361,1),np.ones(361)*-29.5, transform=ccrs.PlateCarree(), color="k")
    plt.title("Sample Locations ("+str(len(dataArray["lat"]))+")")

def SAMLatPlot (plotNo):
    '''Plots SAM index latitudes on a map, requires selLat40 and selLat65'''
    plt.figure(plotNo, figsize=(7,7))
    ax = plt.axes(projection=ccrs.SouthPolarStereo())
    ax.add_feature(cfeature.OCEAN)
    ax.add_feature(cfeature.COASTLINE)
    ax.coastlines()
    ax.gridlines()
    plt.plot(np.arange(0,361,1),np.ones(361)*selLat40, transform=ccrs.PlateCarree(), color="k")
    plt.plot(np.arange(0,361,1),np.ones(361)*selLat65, transform=ccrs.PlateCarree(), color="k")
    plt.title("Selected latitudes for SAM index calculation")

def timePlot (dataArray, runId, label, title, plotNo):
    '''Displays SAM index over time for any given range.'''
    plt.figure(plotNo, figsize = (20,10))
    indexName = "SAM_Index_"+str(runId)
    plt.plot(dataArray["time"], dataArray[indexName], label = str(label))
    #plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.axhline(0, color="Black", lw=0.5)
    plt.legend(loc='upper right')
    plt.ylabel("SAM Index")
    plt.title(str(title))

def timePlotStd (dataSetMean, dataSetError, title, plotNo):
    '''Displays SAM index with standard deviations over a 12 month period.'''
    plt.figure(plotNo, figsize = (10,10))
    plt.errorbar(months, dataSetMean.values, yerr=dataSetError.values, label = dataSetMean.name)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xlabel("Month")
    plt.ylabel("SAM Index")
    plt.title(str(title))

def timePlotYear (dataArray, title, plotNo):
    '''Displays SAM index for a 12 month period starting in January. Compatible with timePlot.'''
    plt.figure(plotNo, figsize = (10,10))
    plt.plot(months, dataArray.values, label = dataArray.name)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.xlabel("Month")
    plt.ylabel("SAM Index")
    plt.title(str(title))

```

```

def timePlotDecade(dataArray, runId, label, title, plotNo):
    '''Displays SAM index over a decade.'''
    timeAxis = np.arange(np.datetime64("2001-01"), np.datetime64("2011-01"), np.timedelta64(1, "D"))
    indexName = "SAM_Index_" + str(runId)
    plt.figure(plotNo, figsize = (20,10))
    plt.plot(timeAxis, dataArray[indexName], label = str(label))
    plt.axhline(0, color="Black", lw=0.5)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.ylabel("SAM Index")
    plt.xlabel("Time")
    plt.title(str(title))

print("Plotting functions defined")

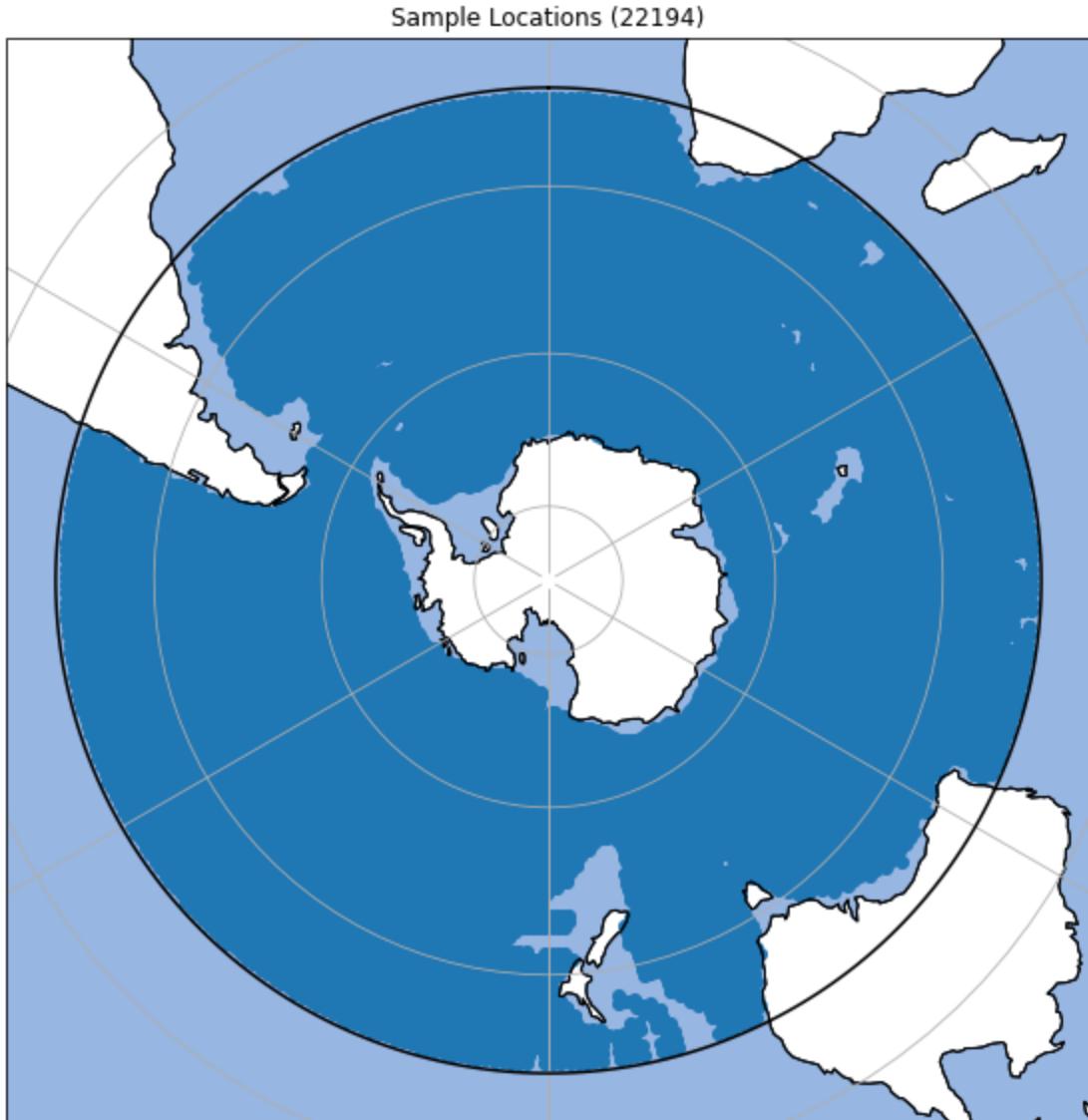
```

Plotting functions defined

Plotting the mask

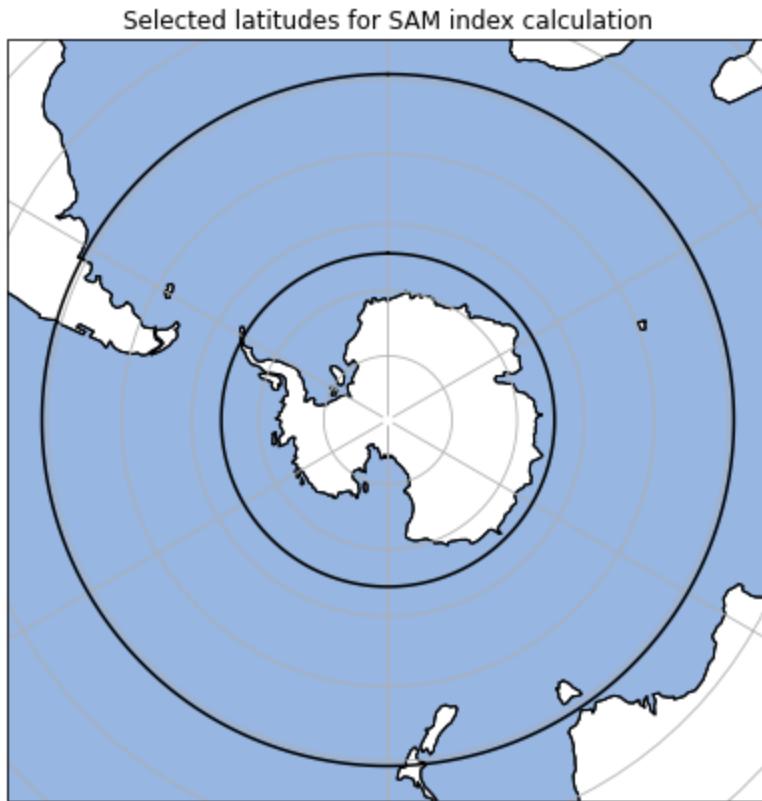
```
In [11]: locationPlotXr(geoRangeFilt, (10,10), 1) #OceanMaskVolcello
print("Ocean Mask from Volcello Data")
plt.show()
```

Ocean Mask from Volcello Data



In [12]:

```
SAMLatPlot(1)  
plt.show()
```



Data Analysis

SAM trend for 1980 - 2009

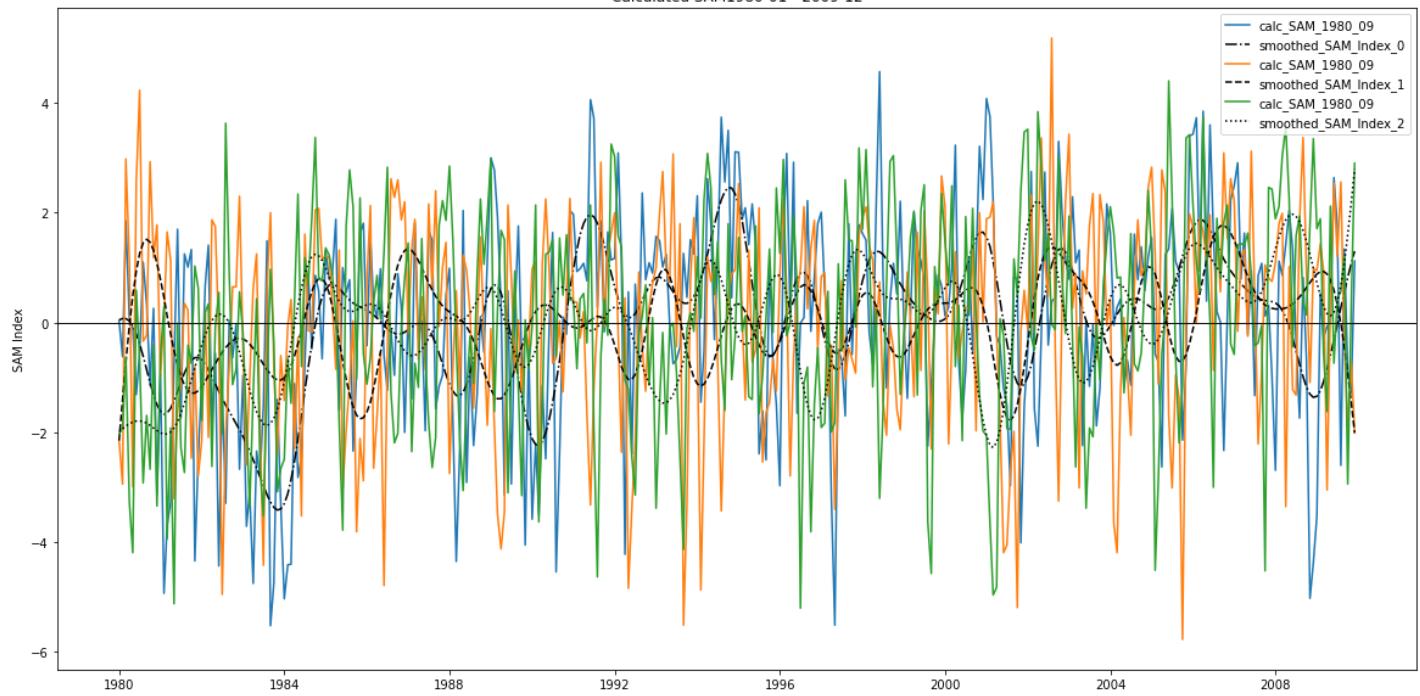
In [13]:

```
startDate, endDate = "1980-01", "2009-12"  
SAM1980_09DF = pd.DataFrame({'time' : dfESMLat.time.sel(time=slice(startDate, endDate)) .va  
  
for i in dfESMLat["RunId"]:  
    SAM1980_09Part_data = dfESMLat.sel(time=slice(startDate, endDate), RunId=i)  
    SAM1980_09Part = SAMCalculateNorm(SAM1980_09Part_data, i)  
    SAM1980_09DF = SAM1980_09DF.merge(SAM1980_09Part, left_on="time", right_on="time")  
  
print("External and internal SAM calculated for "+startDate+" - "+endDate+".")
```

External and internal SAM calculated for 1980-01 - 2009-12.

In [14]:

```
smooth18SAM1980_09Arr = np.empty(shape=(0, len(SAM1980_09DF["time"])))  
for i in range(fileCount):  
    indexName = "SAM_Index_"+str(i)  
    smooth18SAM1980_09 = butter_lowpass(SAM1980_09DF[indexName], 18)  
    smooth18SAM1980_09Arr = np.append(smooth18SAM1980_09Arr, [smooth18SAM1980_09], axis=0)  
  
timePlot(SAM1980_09DF, i, "calc_SAM_1980_09", "Calculated SAM"+startDate+" - "+endDate)  
plt.plot(SAM1980_09DF["time"], smooth18SAM1980_09Arr[i%3], color="Black", ls=lineType  
plt.axhline(0, color="Black", lw=0.5)  
plt.legend()  
plt.show()
```



SAM smoothing

In [15]:

```
startDate, endDate = "1980-01", "2009-12"
extSAM1980_09 = externalSAMSearch(startDate, endDate)
print("External SAM calculated for "+startDate+" - "+endDate+".)
```

External SAM calculated for 1980-01 - 2009-12.

In [16]:

```
smooth18SAM1980_09Arr = np.empty(shape=(0, len(SAM1980_09DF["time"])))
smooth24SAM1980_09Arr = np.empty(shape=(0, len(SAM1980_09DF["time"])))
smooth120SAM1980_09Arr = np.empty(shape=(0, len(SAM1980_09DF["time"])))

smooth18extSAM1980_09 = butter_lowpass(extSAM1980_09[1], 18)
for i in range(fileCount):
    indexName = "SAM_Index_"+str(i)
    smooth18SAM1980_09 = butter_lowpass(SAM1980_09DF[indexName], 18)
    smooth18SAM1980_09Arr = np.append(smooth18SAM1980_09Arr, [smooth18SAM1980_09], axis=0)

smooth24extSAM1980_09 = butter_lowpass(extSAM1980_09[1], 24)
for i in range(fileCount):
    indexName = "SAM_Index_"+str(i)
    smooth24SAM1980_09 = butter_lowpass(SAM1980_09DF[indexName], 24)
    smooth24SAM1980_09Arr = np.append(smooth24SAM1980_09Arr, [smooth24SAM1980_09], axis=0)

smooth120extSAM1980_09 = butter_lowpass(extSAM1980_09[1], 120)
for i in range(fileCount):
    indexName = "SAM_Index_"+str(i)
    smooth120SAM1980_09 = butter_lowpass(SAM1980_09DF[indexName], 120)
    smooth120SAM1980_09Arr = np.append(smooth120SAM1980_09Arr, [smooth120SAM1980_09], axis=0)

print("Smoothing calculated for SAM Index.")
```

Smoothing calculated for SAM Index.

In [17]:

```
plt.figure(1, figsize=(20,10))
plt.plot(extSAM1980_09[0], extSAM1980_09[1], label="external_SAM_1980_09")
plt.plot(SAM1980_09DF["time"], smooth18extSAM1980_09, color="Black", label="external_18_SAM1980_09")
for i in range(fileCount):
```

```

timePlot(SAM1980_09DF, i, "calc_SAM_1980_09"+str(i), "Calculated vs External SAM "+str(i))
plt.plot(SAM1980_09DF["time"], smooth18SAM1980_09Arr[i], color="Black", ls=lineType[i])
plt.legend()

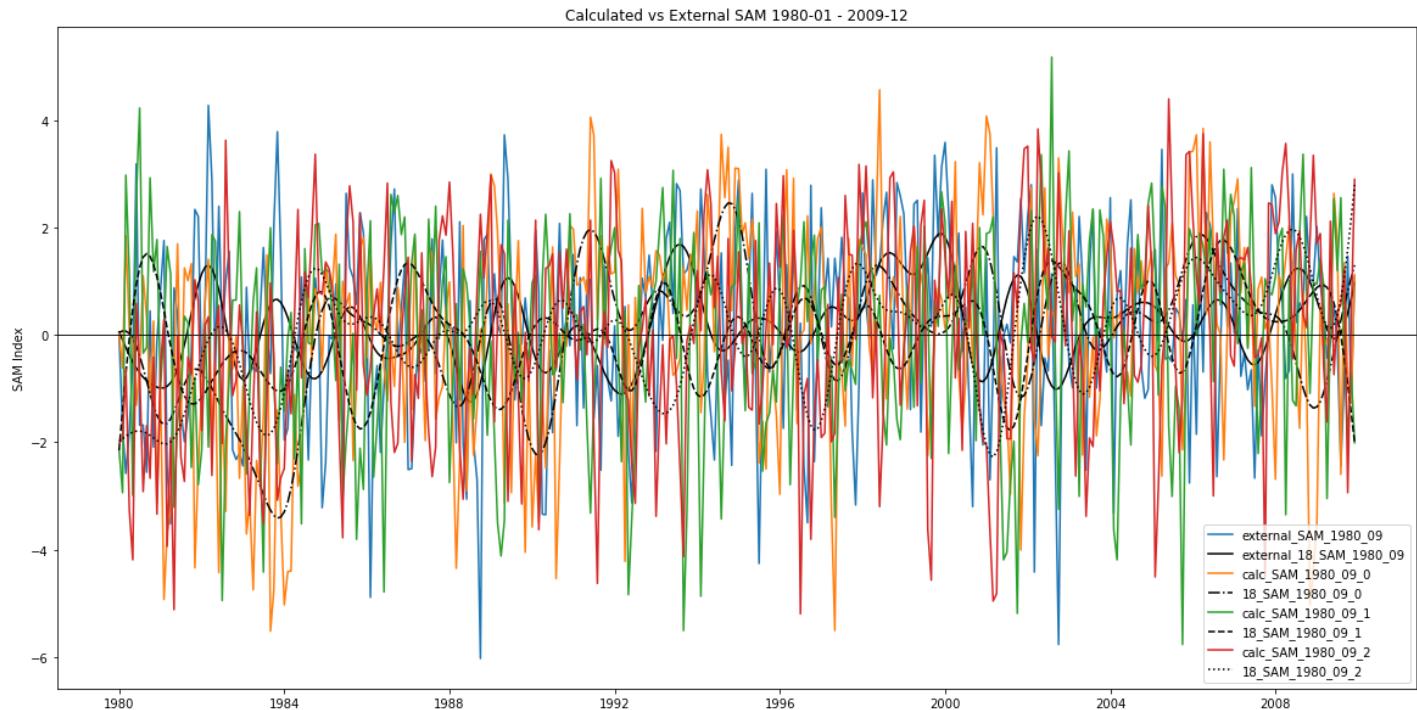
plt.figure(2, figsize=(20,10))
plt.plot(SAM1980_09DF["time"], smooth18extSAM1980_09, color="Blue", label="external_18_SAM")
for i in range(fileCount):
    plt.plot(SAM1980_09DF["time"], smooth18SAM1980_09Arr[i], label="18_SAM_1980_09"+str(i))
plt.legend()

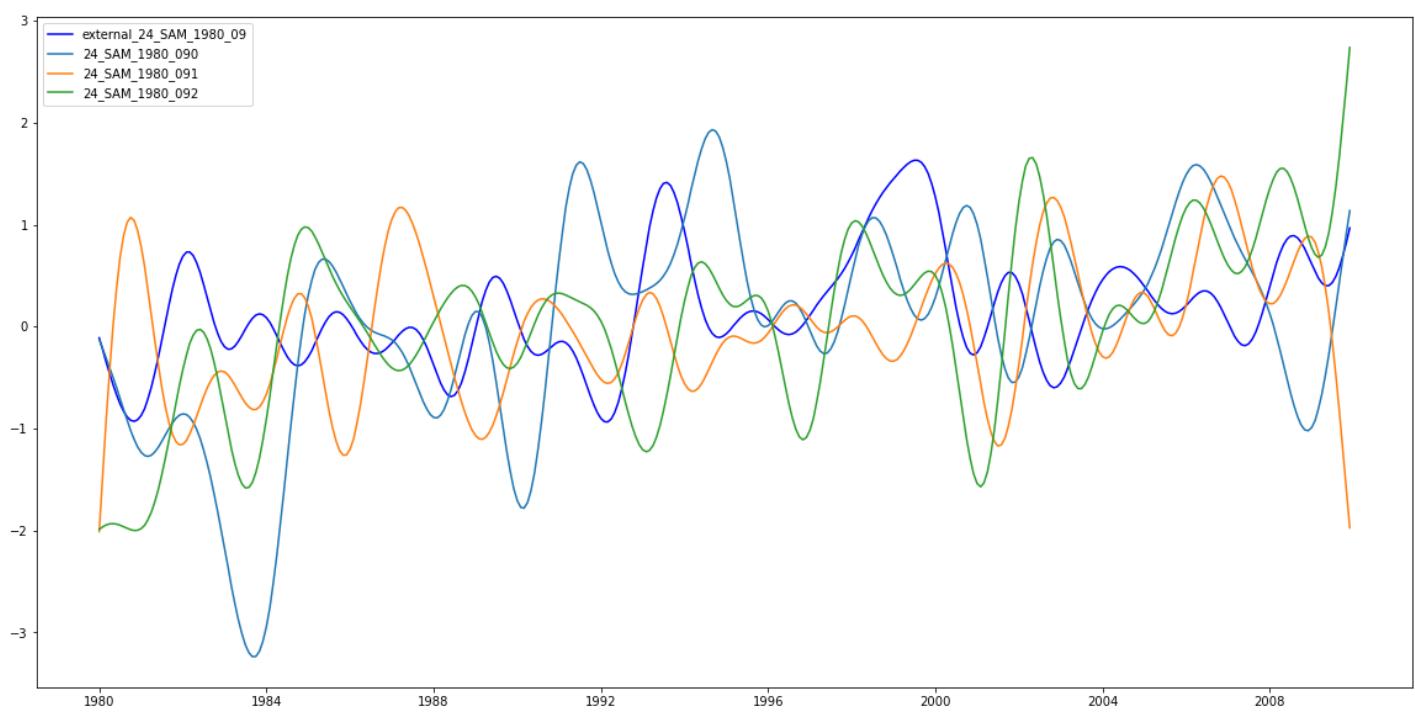
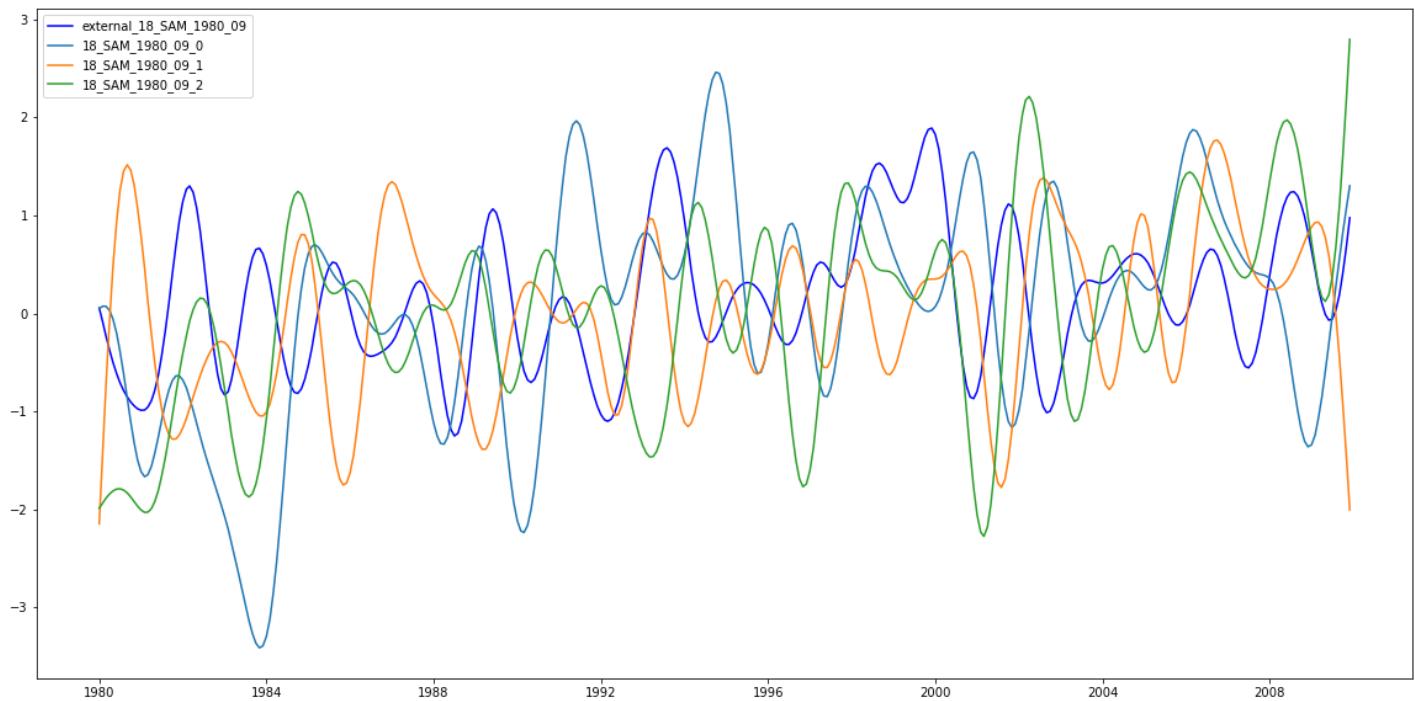
plt.figure(3, figsize=(20,10))
plt.plot(SAM1980_09DF["time"], smooth24extSAM1980_09, color="Blue", label="external_24_SAM")
for i in range(fileCount):
    plt.plot(SAM1980_09DF["time"], smooth24SAM1980_09Arr[i], label="24_SAM_1980_09"+str(i))
plt.legend()

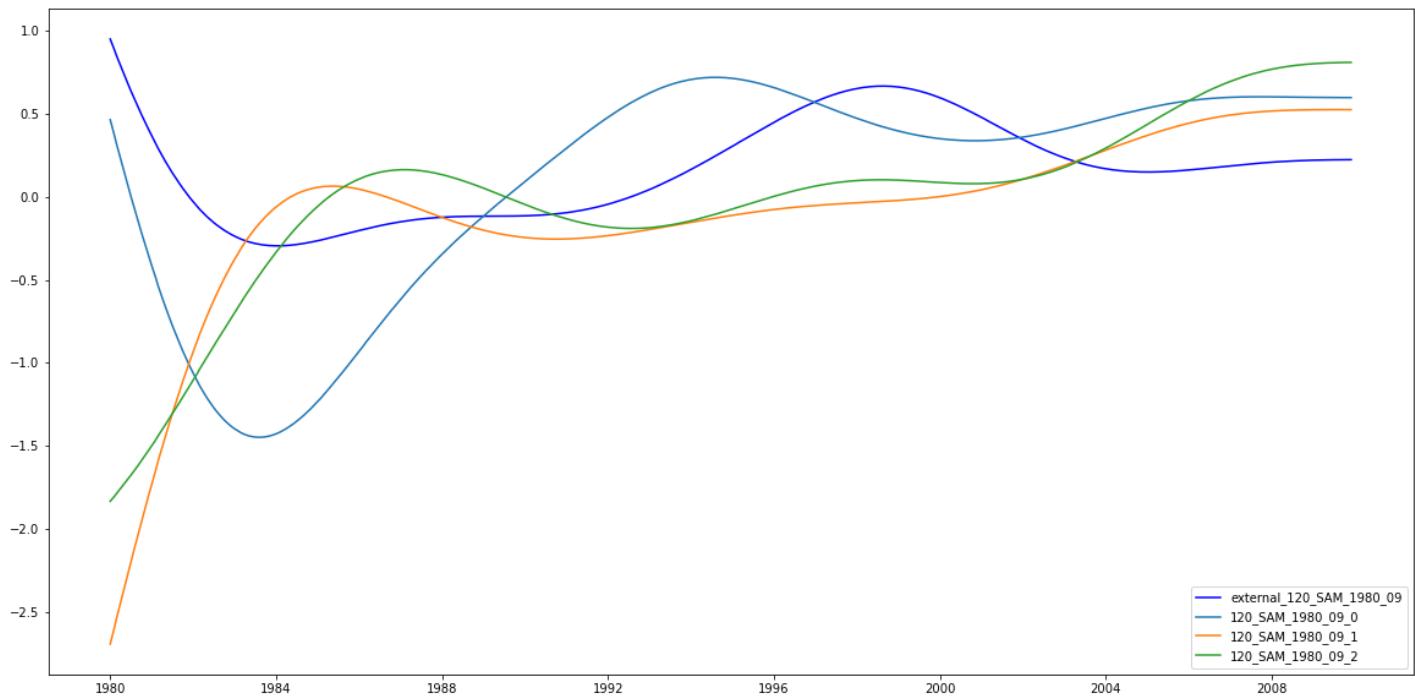
plt.figure(4, figsize=(20,10))
plt.plot(SAM1980_09DF["time"], smooth120extSAM1980_09, color="Blue", label="external_120_SAM")
for i in range(fileCount):
    plt.plot(SAM1980_09DF["time"], smooth120SAM1980_09Arr[i], label="120_SAM_1980_09"+str(i))
plt.legend()

plt.show()

```







Mean Latitude Data

In [18]:

```
smoothLatArrLen = len(LatMeanList[0][0])
smooth18Lat1980_09Arr = np.empty(shape=(0, 4, smoothLatArrLen))
smooth24Lat1980_09Arr = np.empty(shape=(0, 4, smoothLatArrLen))
smooth50Lat1980_09Arr = np.empty(shape=(0, 4, smoothLatArrLen))

for i in range(len(GMModelNames)):
    currentLatMeans = LatMeanList[i]
    smooth18TempArr = np.empty(shape=(0, smoothLatArrLen))
    smooth24TempArr = np.empty(shape=(0, smoothLatArrLen))
    smooth50TempArr = np.empty(shape=(0, smoothLatArrLen))
    for j in range(len(thresholds)):
        currentLatMeanSingle = currentLatMeans[j].squeeze()
        smooth18TempArr = np.append(smooth18TempArr, [butter_lowpass(currentLatMeanSingle,
        smooth24TempArr = np.append(smooth24TempArr, [butter_lowpass(currentLatMeanSingle,
        smooth50TempArr = np.append(smooth50TempArr, [butter_lowpass(currentLatMeanSingle,

smooth18Lat1980_09Arr = np.append(smooth18Lat1980_09Arr, [smooth18TempArr], axis=0)
smooth24Lat1980_09Arr = np.append(smooth24Lat1980_09Arr, [smooth24TempArr], axis=0)
smooth50Lat1980_09Arr = np.append(smooth50Lat1980_09Arr, [smooth50TempArr], axis=0)
print("Smoothing calculated for SAM Index.")
```

Smoothing calculated for SAM Index.

In [19]:

```
thresholdsLen = len(thresholds)
currSmooth18MeanArr = np.empty(shape=(0, smoothLatArrLen))
currSmooth24MeanArr = np.empty(shape=(0, smoothLatArrLen))
currSmooth50MeanArr = np.empty(shape=(0, smoothLatArrLen))

for i in range(thresholdsLen):
    currSmooth18Mean = np.mean(smooth18Lat1980_09Arr[:,i], axis=0)
    currSmooth24Mean = np.mean(smooth24Lat1980_09Arr[:,i], axis=0)
    currSmooth50Mean = np.mean(smooth50Lat1980_09Arr[:,i], axis=0)

    currSmooth18MeanArr = np.append(currSmooth18MeanArr, [currSmooth18Mean], axis=0)
    currSmooth24MeanArr = np.append(currSmooth24MeanArr, [currSmooth24Mean], axis=0)
    currSmooth50MeanArr = np.append(currSmooth50MeanArr, [currSmooth50Mean], axis=0)

print("Smoothing Averages calculated.")
```

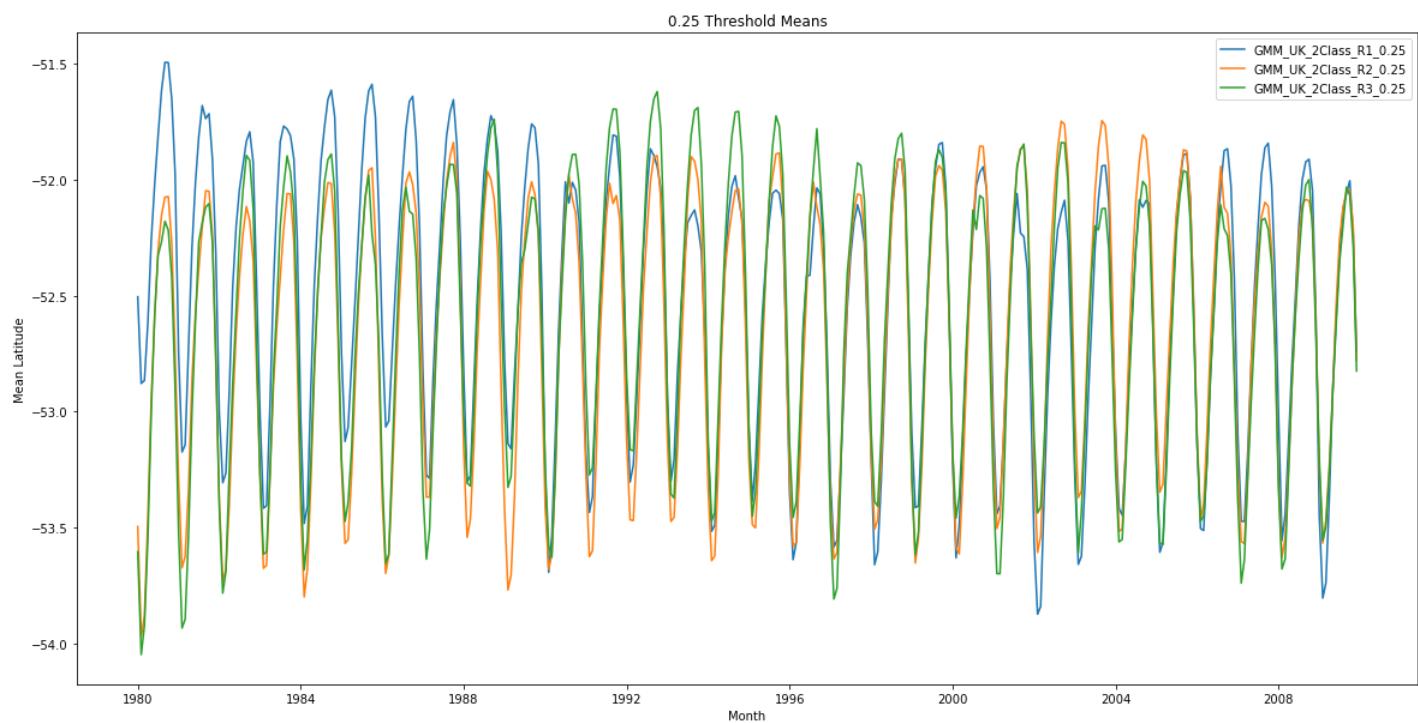
Smoothing Averages calculated.

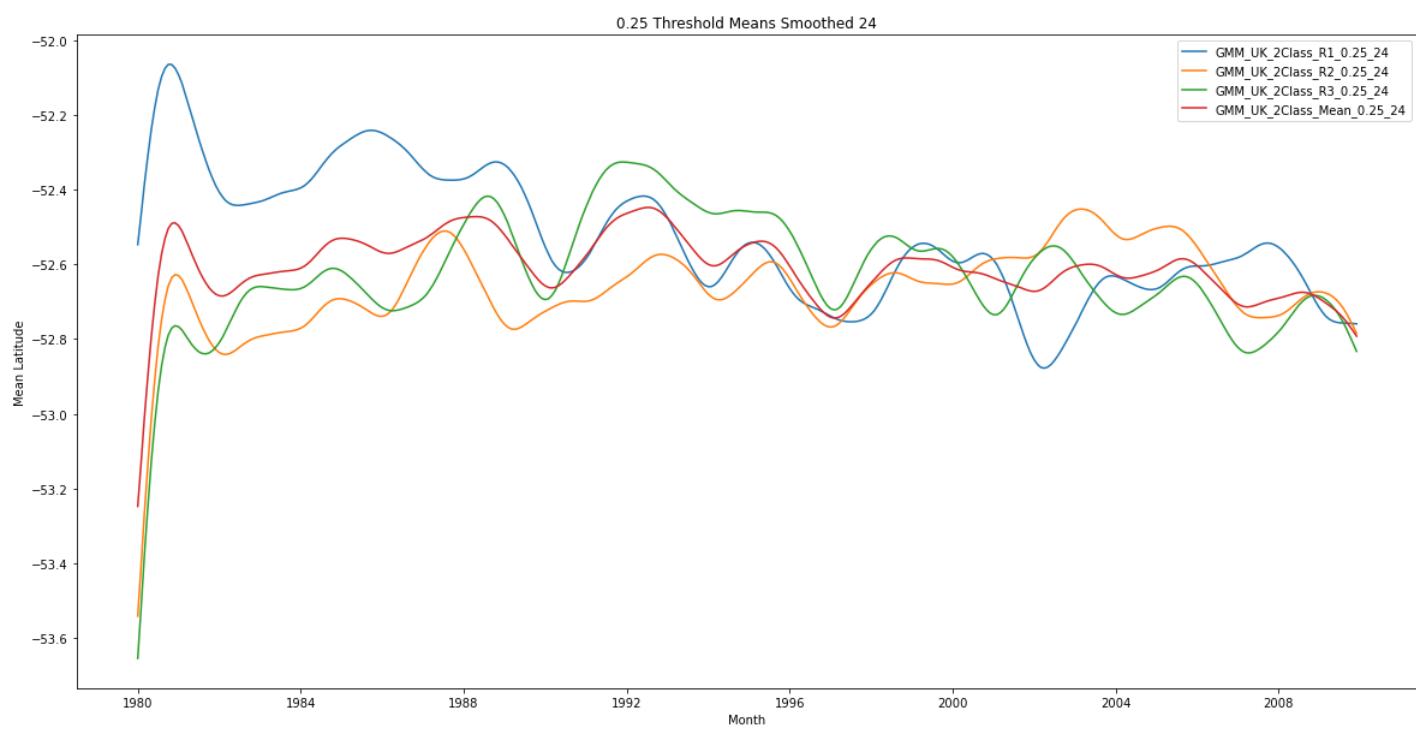
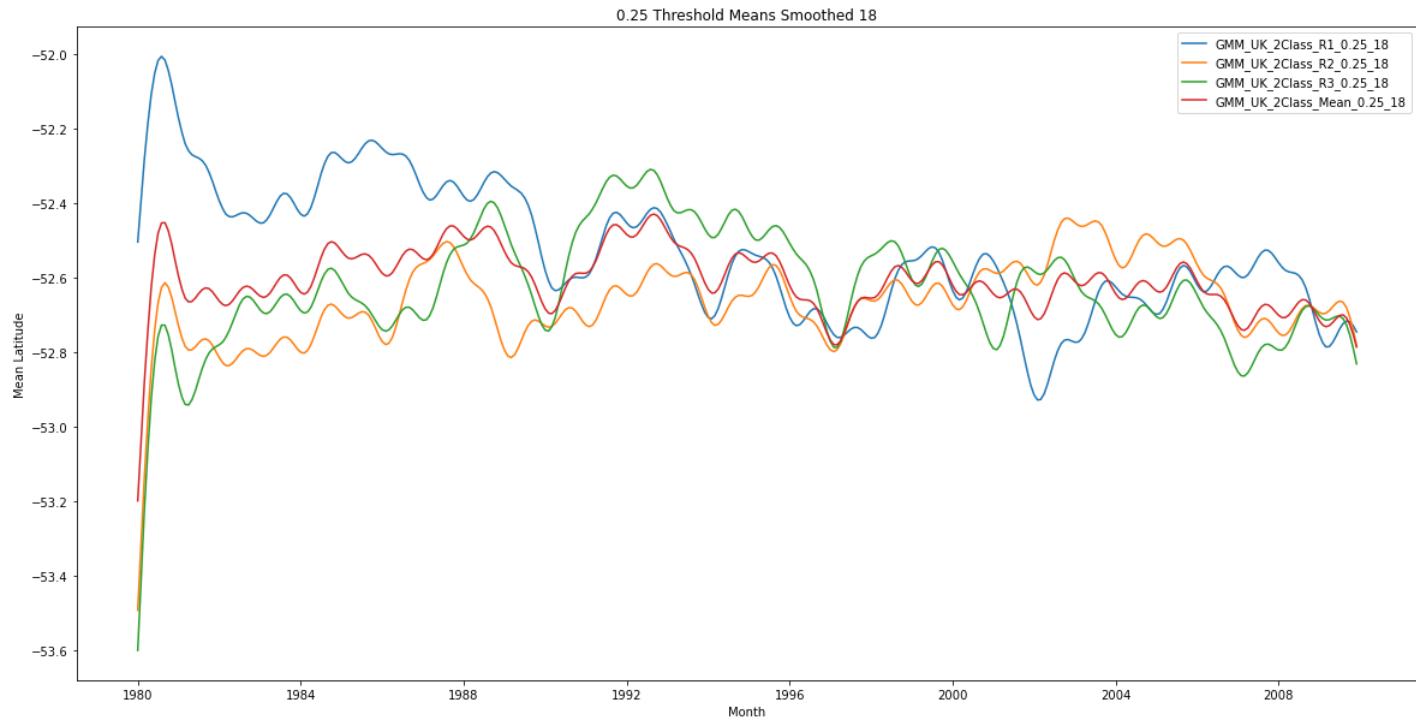
In [20]:

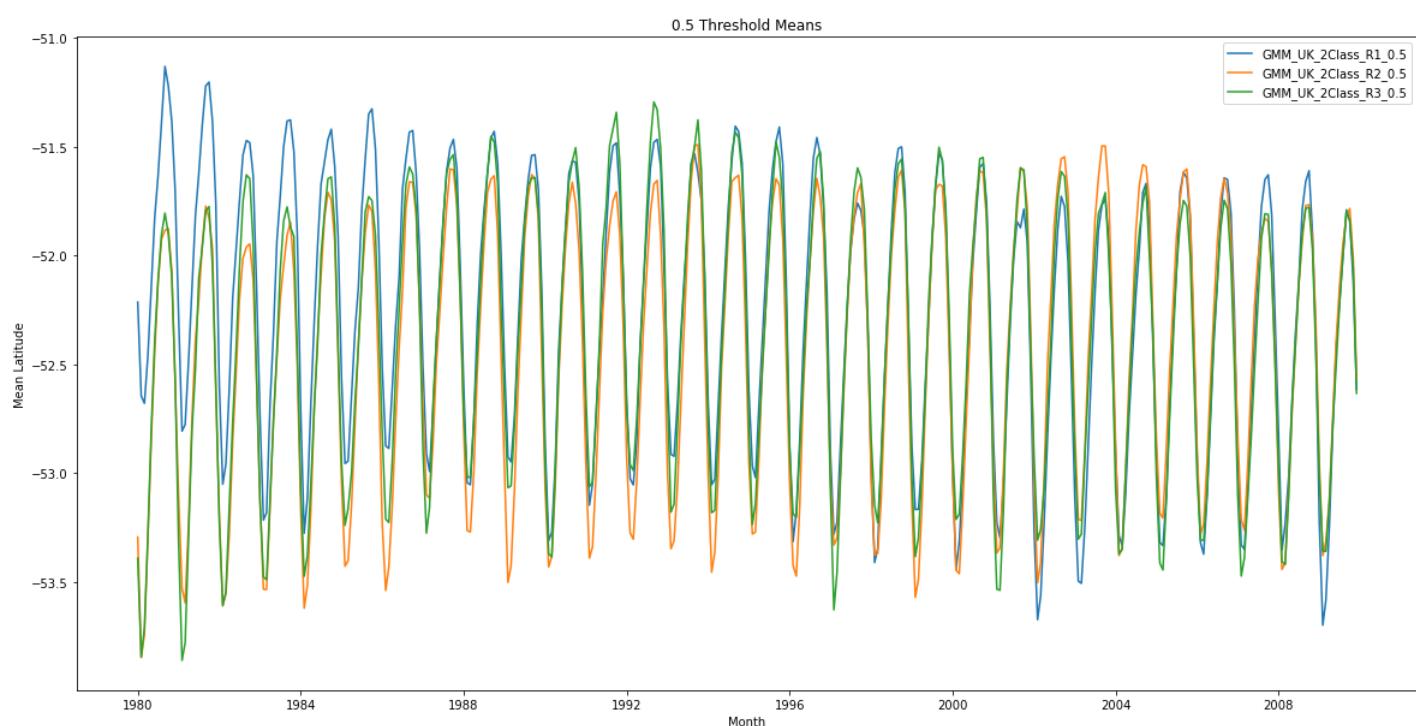
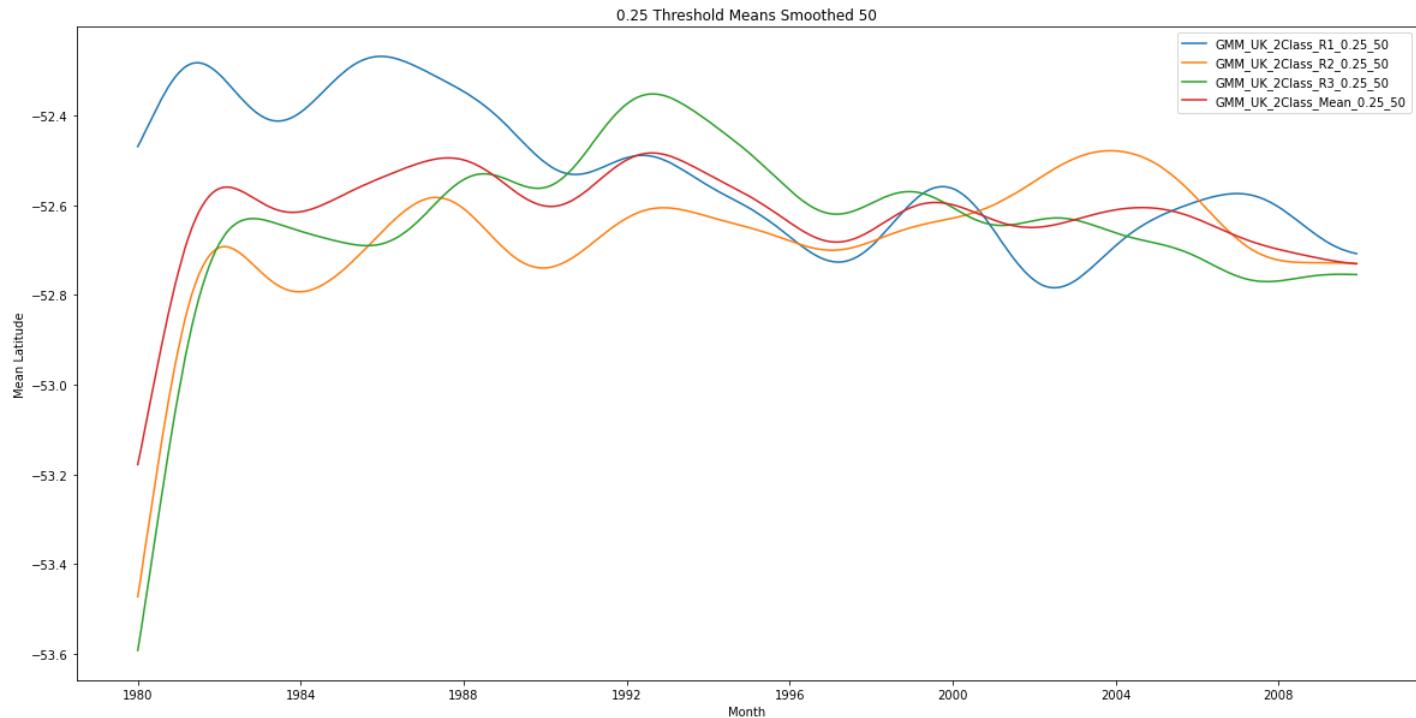
```
thresholdsLen = len(thresholds)
for i in range(len(GMModelNames)):
    currentLatMeans = LatMeanList[i]
    currentLatSmooth18 = smooth18Lat1980_09Arr[i]
    currentLatSmooth24 = smooth24Lat1980_09Arr[i]
    currentLatSmooth50 = smooth50Lat1980_09Arr[i]
    for j in range(thresholdsLen):
        latMeanPlot(currentLatMeans[j], str(GMModelNames[i])+"_"+str(thresholds[j]), j)
        latMeanPlot(currentLatSmooth18[j], str(GMModelNames[i])+"_"+str(thresholds[j])+"_18")
        latMeanPlot(currentLatSmooth24[j], str(GMModelNames[i])+"_"+str(thresholds[j])+"_24")
        latMeanPlot(currentLatSmooth50[j], str(GMModelNames[i])+"_"+str(thresholds[j])+"_50")

    for i in range(thresholdsLen):
        latMeanPlot(currSmooth18MeanArr[i], "GMM_UK_2Class_Mean_"+str(thresholds[i])+"_18", i)
        latMeanPlot(currSmooth24MeanArr[i], "GMM_UK_2Class_Mean_"+str(thresholds[i])+"_24", i)
        latMeanPlot(currSmooth50MeanArr[i], "GMM_UK_2Class_Mean_"+str(thresholds[i])+"_50", i)

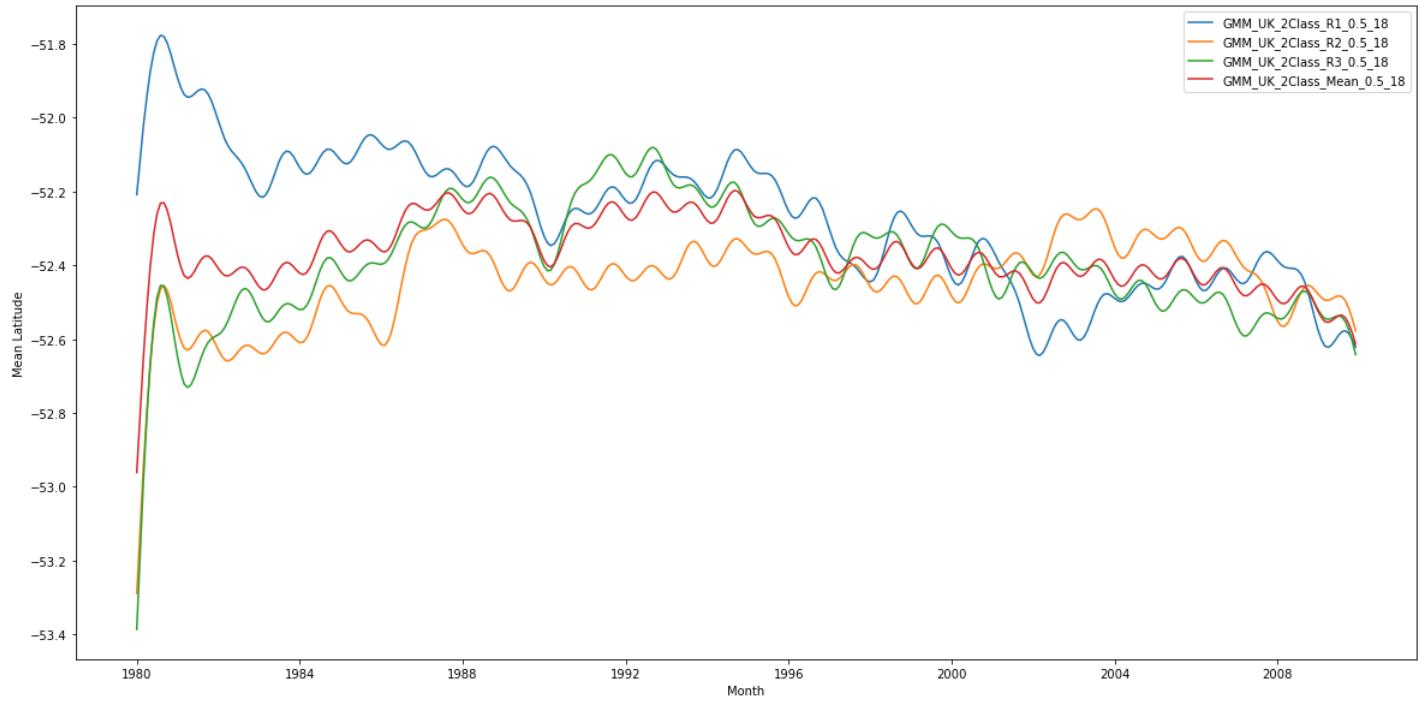
    for j in range(len(thresholds)):
        threshString = str(thresholds[j])
        plt.figure(j)
        plt.title(threshString+" Threshold Means")
        plt.figure(j+thresholdsLen)
        plt.title(threshString+" Threshold Means Smoothed 18")
        plt.figure(j+2*thresholdsLen)
        plt.title(threshString+" Threshold Means Smoothed 24")
        plt.figure(j+3*thresholdsLen)
        plt.title(threshString+" Threshold Means Smoothed 50")
plt.show()
```



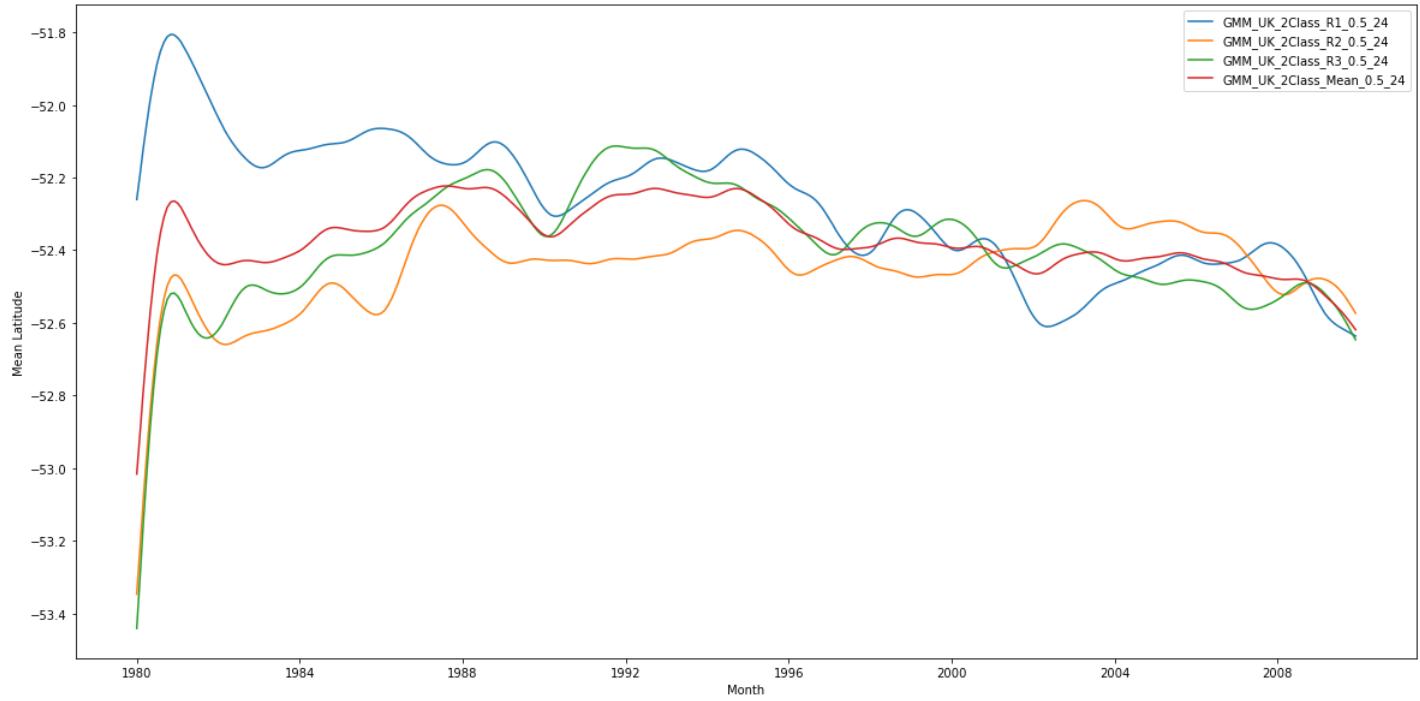


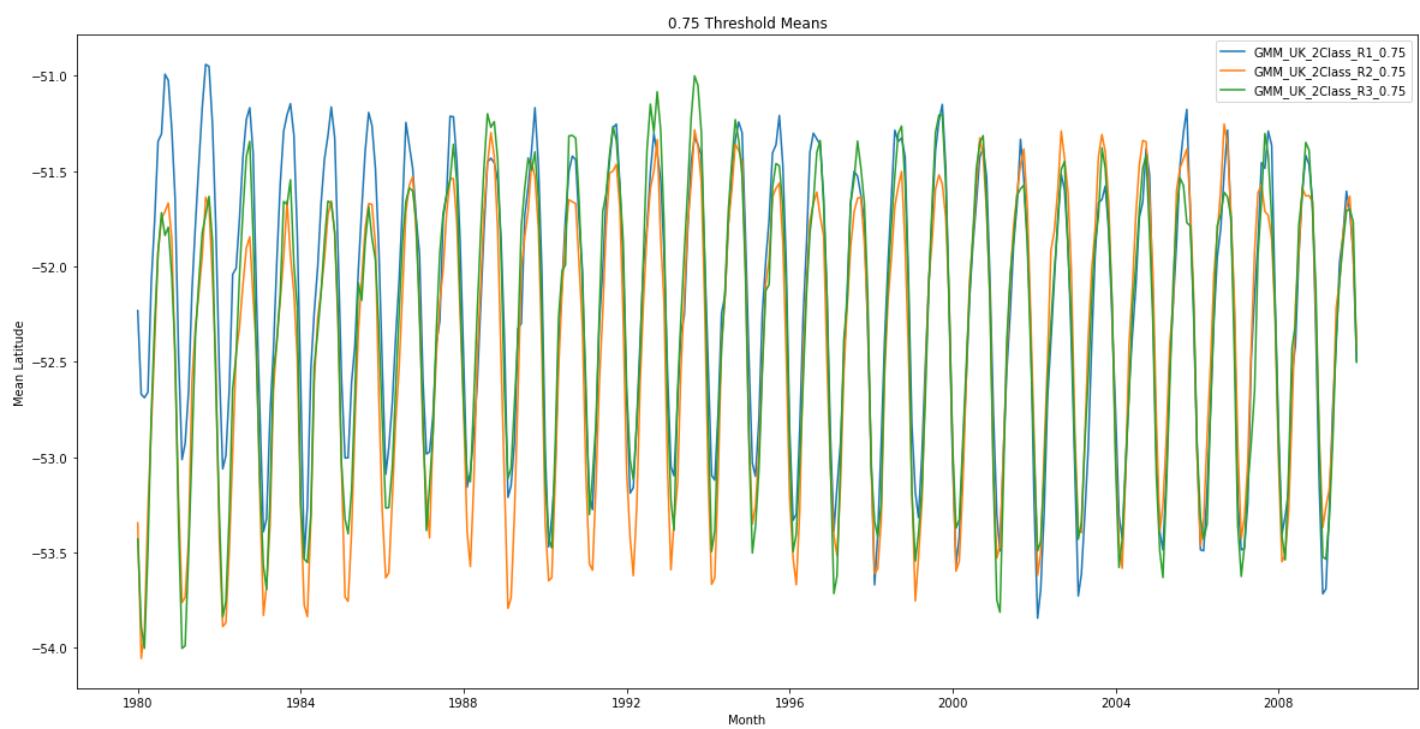
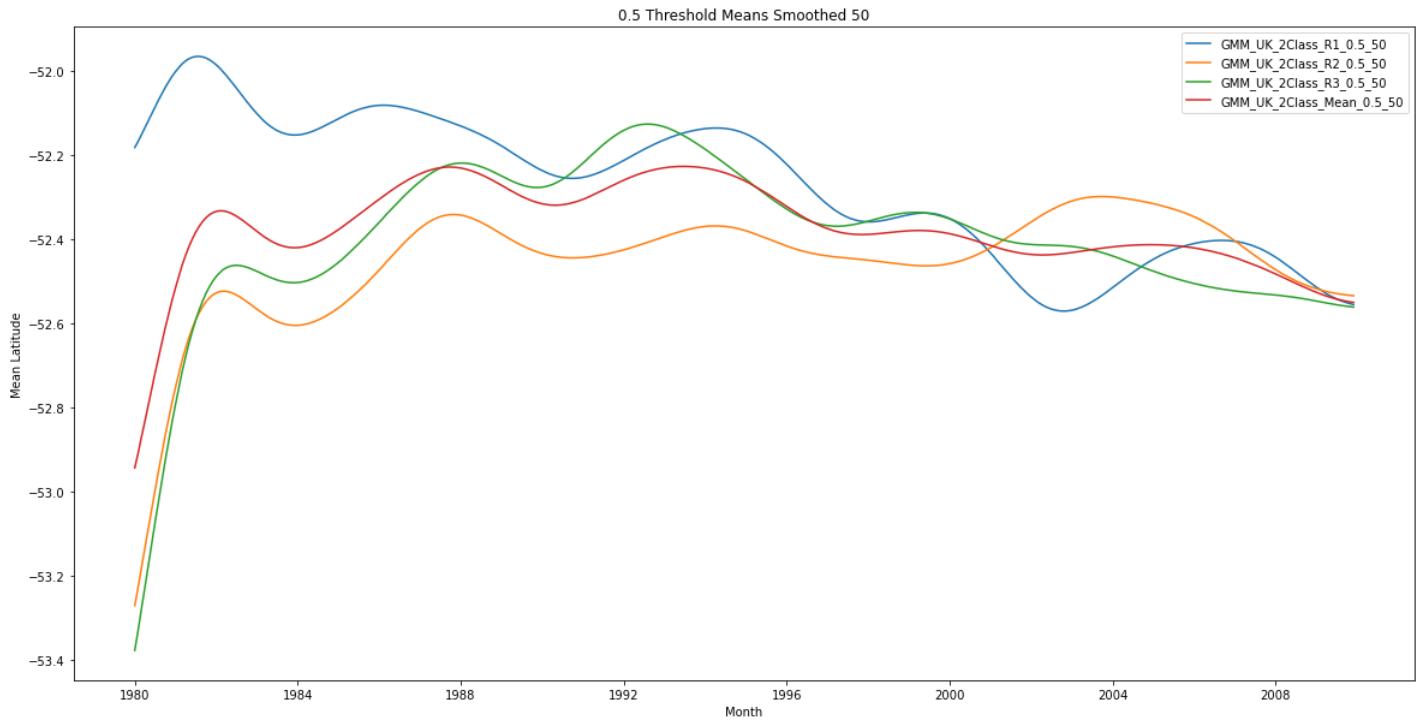


0.5 Threshold Means Smoothed 18

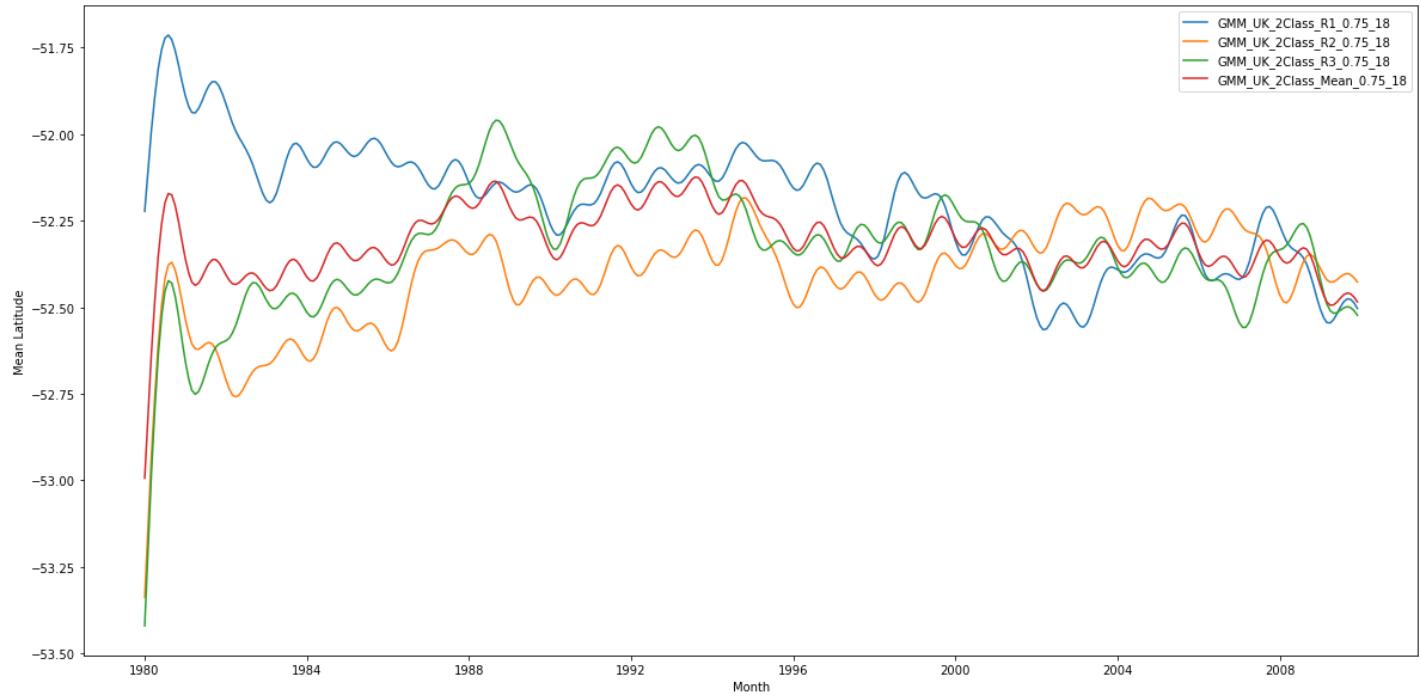


0.5 Threshold Means Smoothed 24

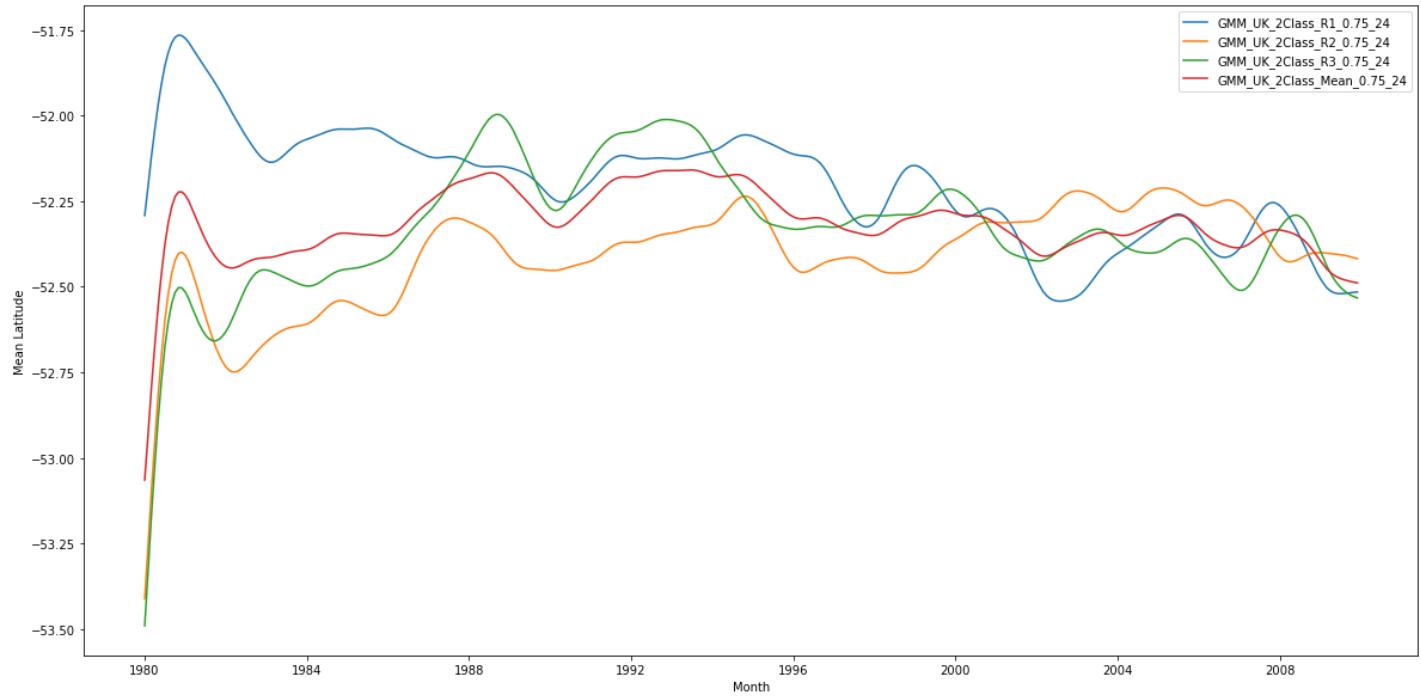


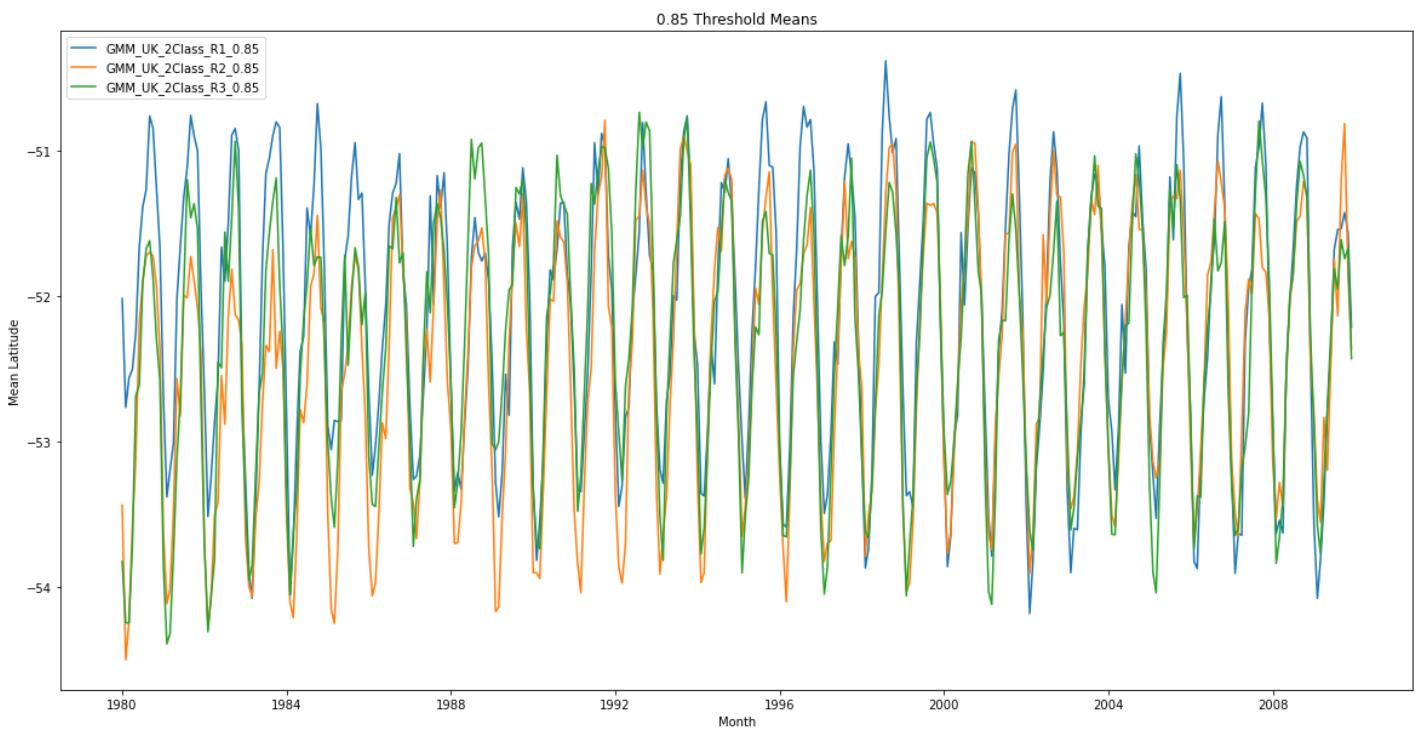
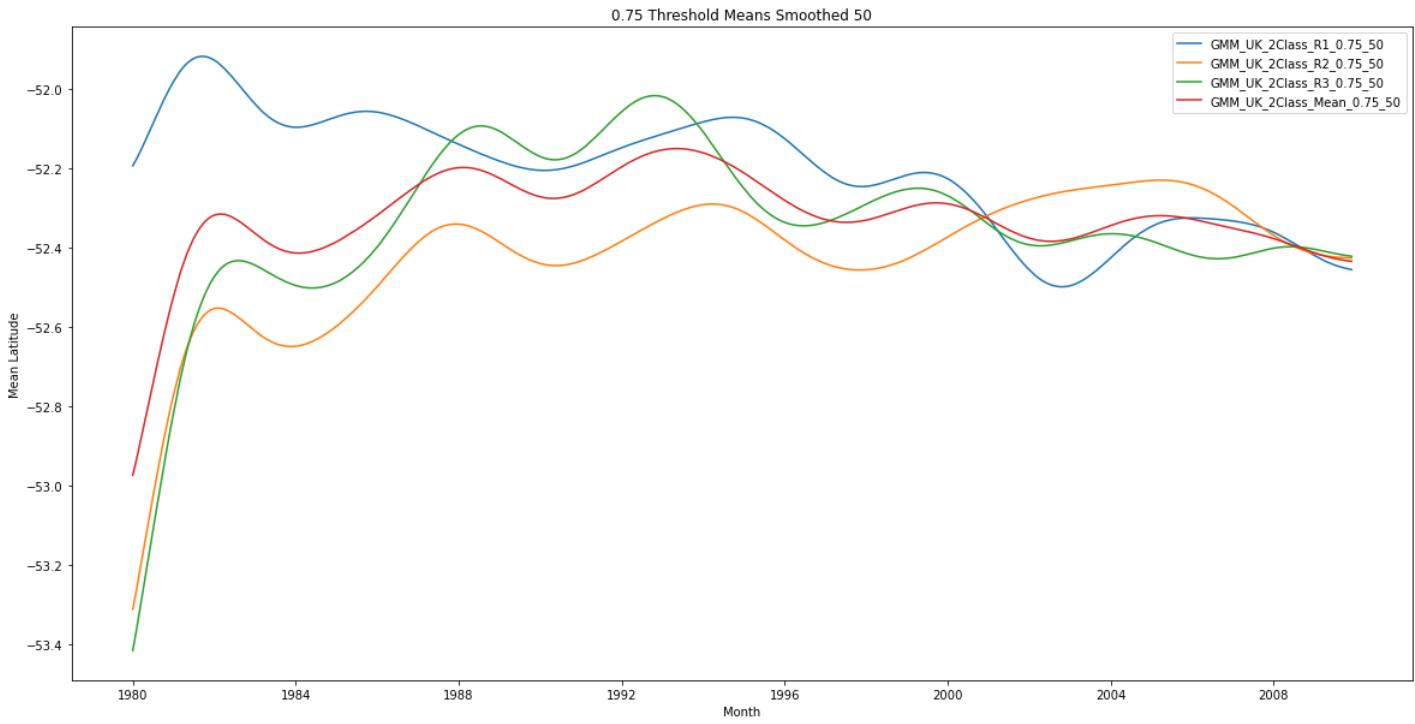


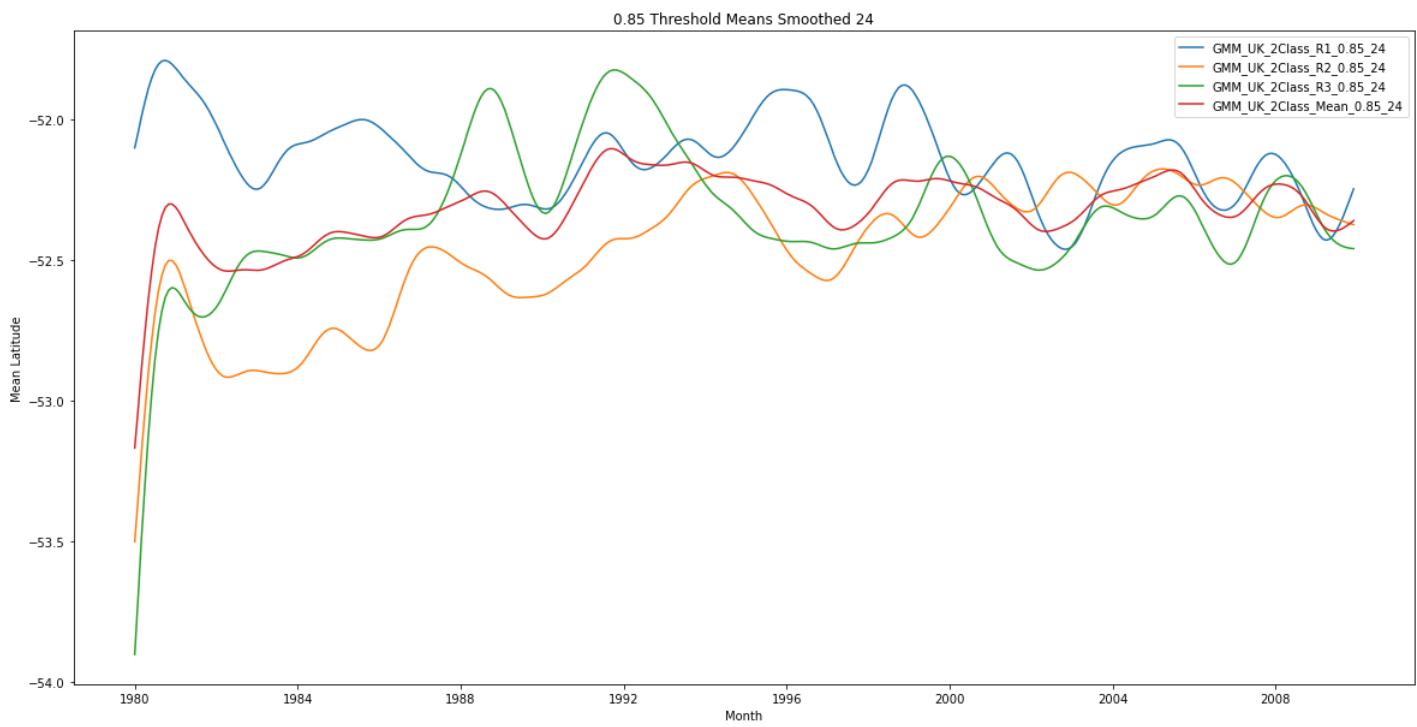
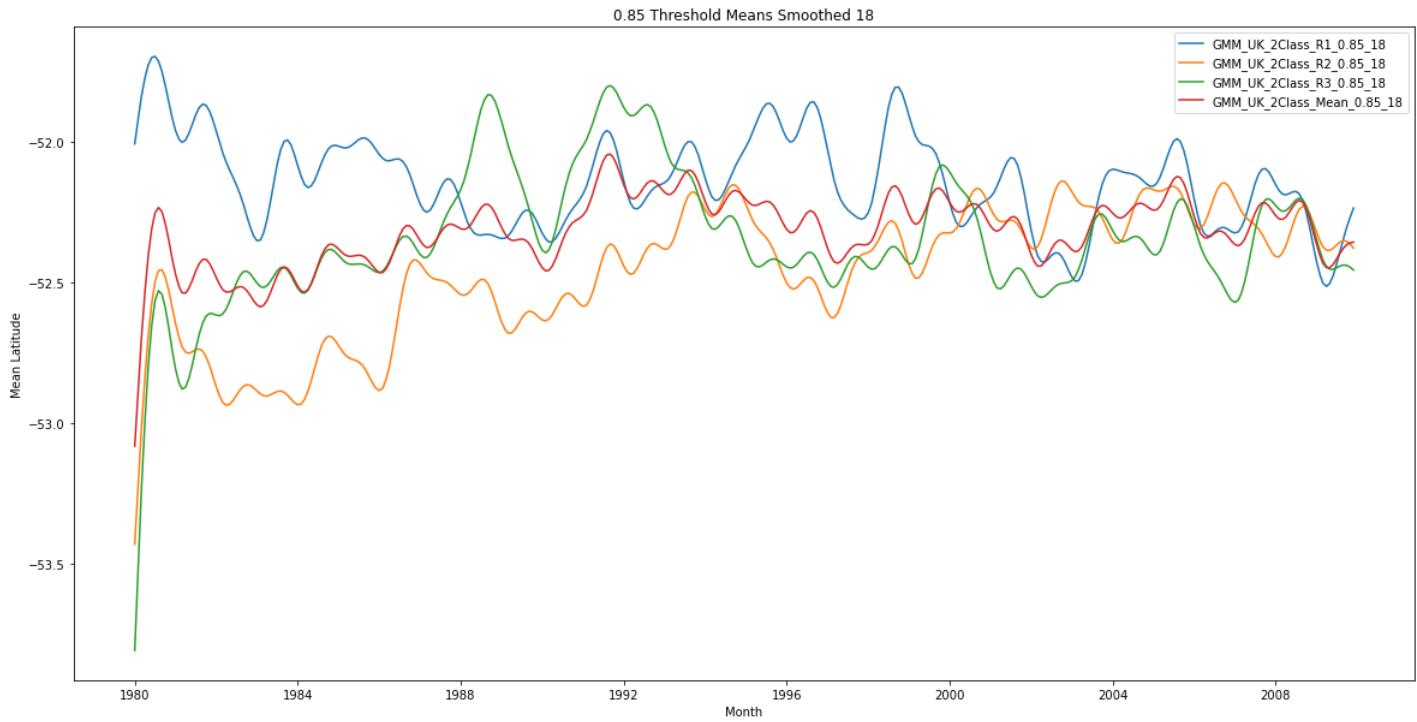
0.75 Threshold Means Smoothed 18

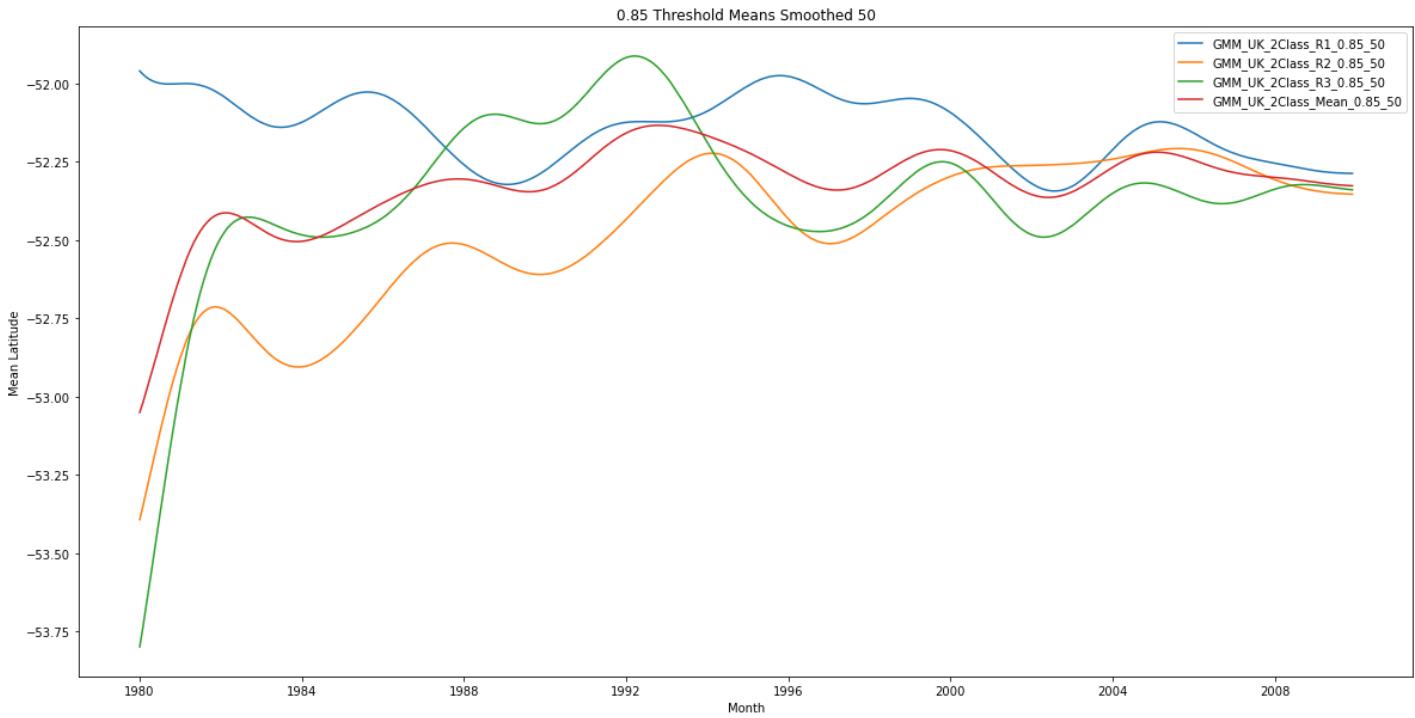


0.75 Threshold Means Smoothed 24









SAM vs Latitude Means

In [21]:

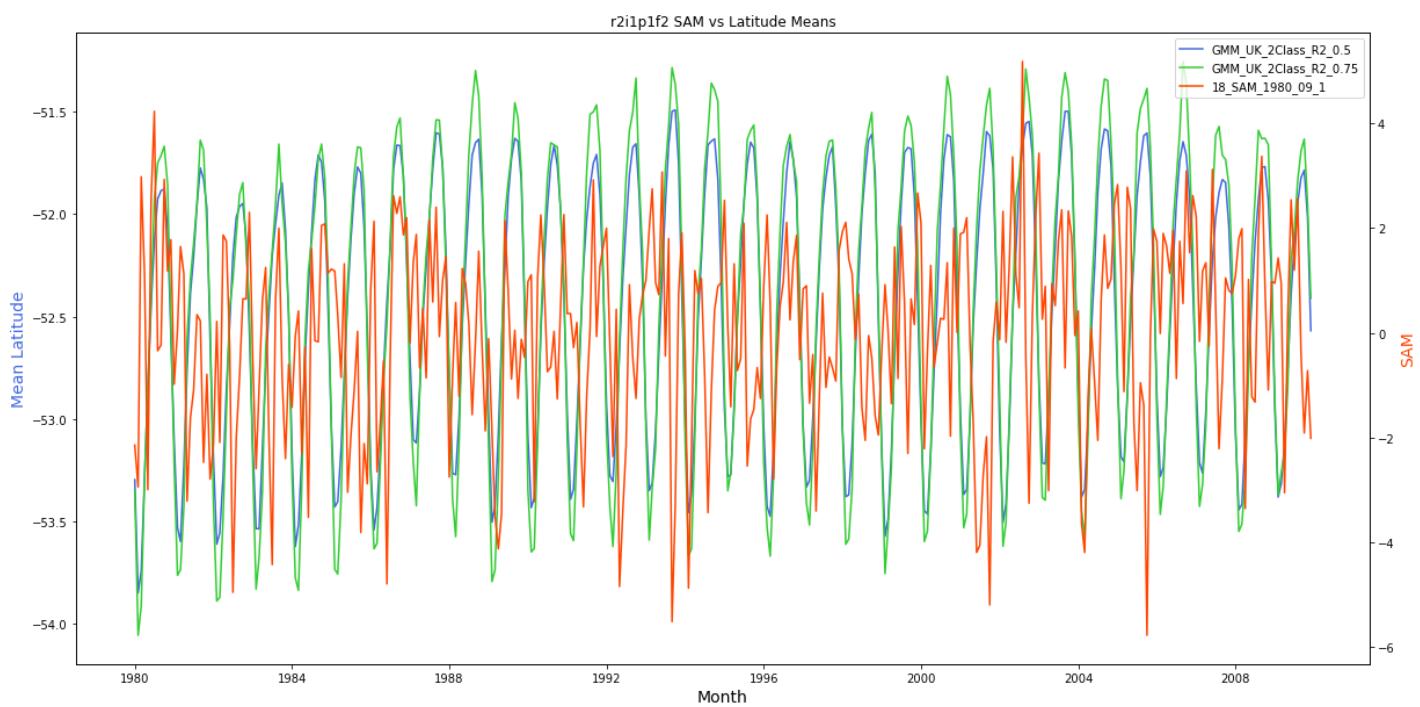
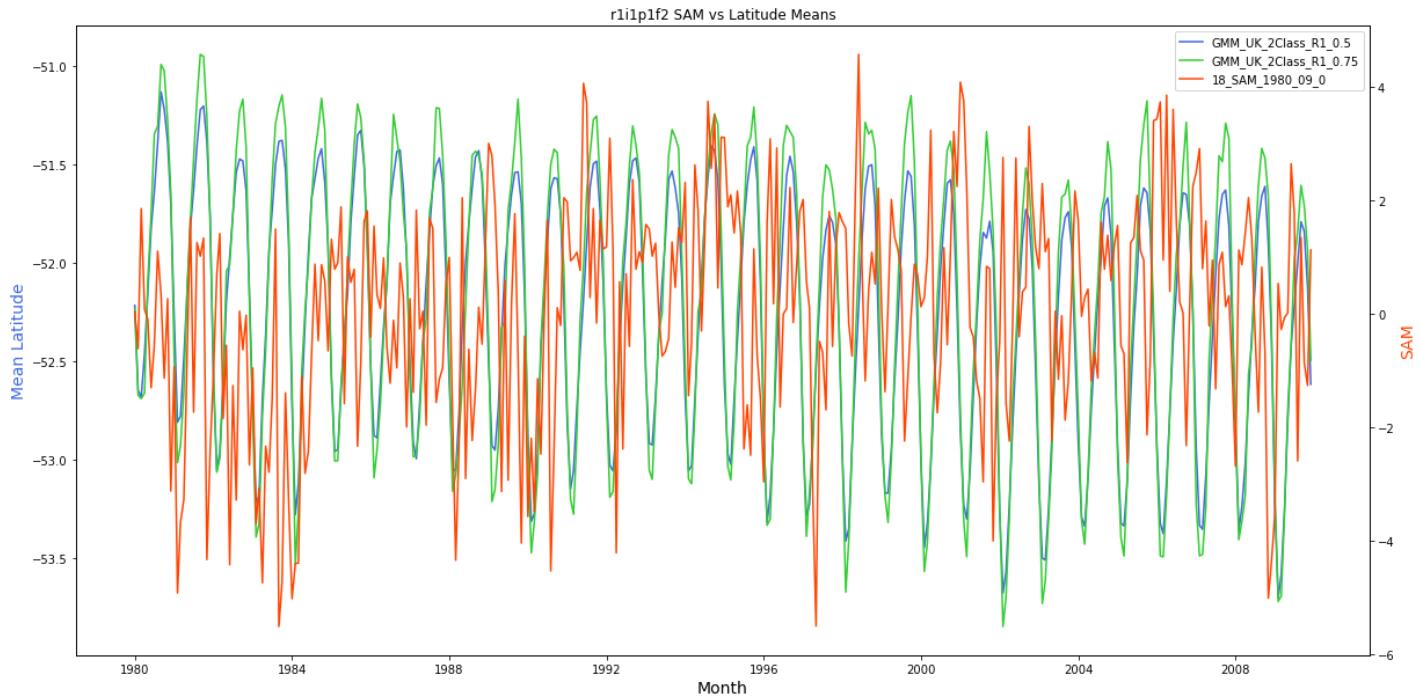
```

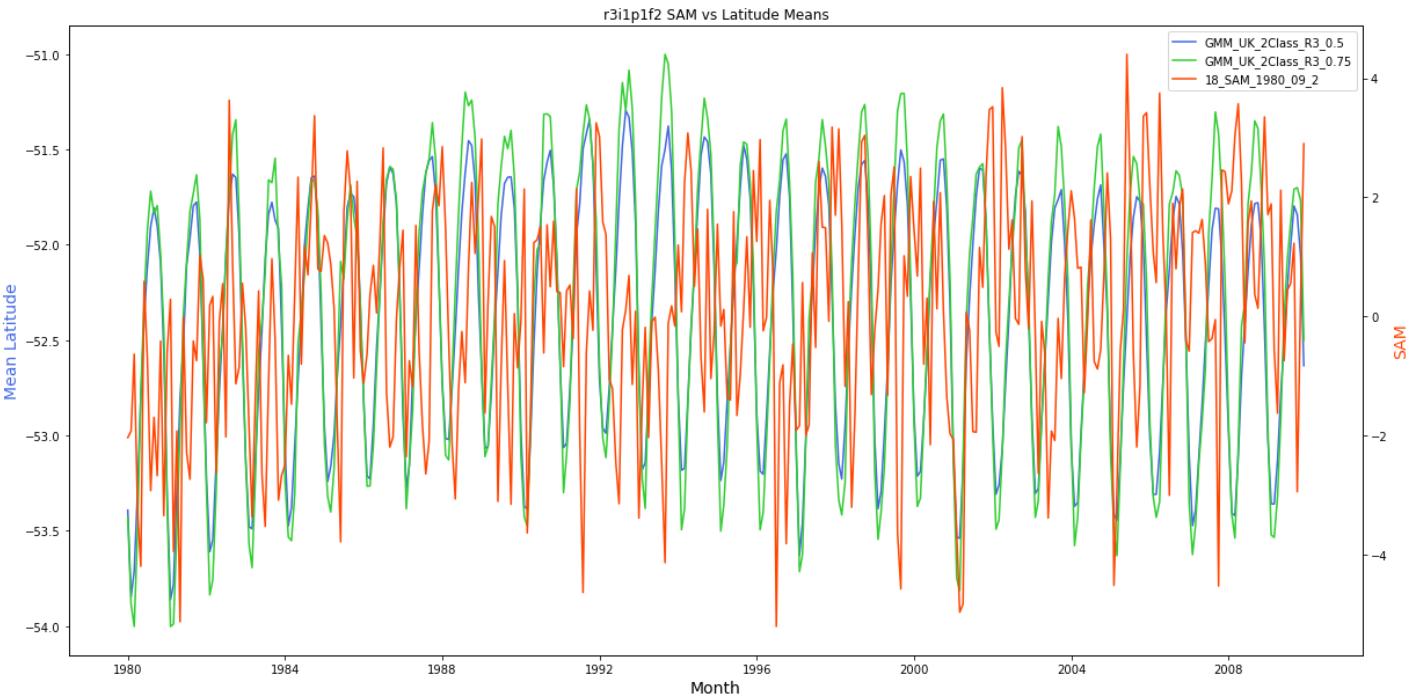
thresholdLook1 = 1
thresholdLook2 = 2
print("SAM vs Latitude Means")
for i in range(fileCount):
    fig,ax = plt.subplots()
    fig.set_figheight(10)
    fig.set_figwidth(20)
    ax.plot(timeAxis, LatMeanList[i][thresholdLook1], label=str(GMMModelNames[i])+"_"+str(t
    ax.plot(timeAxis, LatMeanList[i][thresholdLook2], label=str(GMMModelNames[i])+"_"+str(t
    ax.title.set_text(str(approvedIds[i]+" SAM vs Latitude Means"))

    indexName = "SAM_Index_"+str(i)
    ax2 = ax.twinx()
    ax2.plot(SAM1980_09DF["time"], SAM1980_09DF[indexName], label="18_SAM_1980_09_"+str(i))
    ax.set_xlabel("Month", fontsize=14)
    ax.set_ylabel("Mean Latitude", fontsize=14, color="RoyalBlue")
    ax2.set_ylabel("SAM", fontsize=14, color="OrangeRed")
    lines, labels = ax.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(lines + lines2, labels + labels2)

```

SAM vs Latitude Means





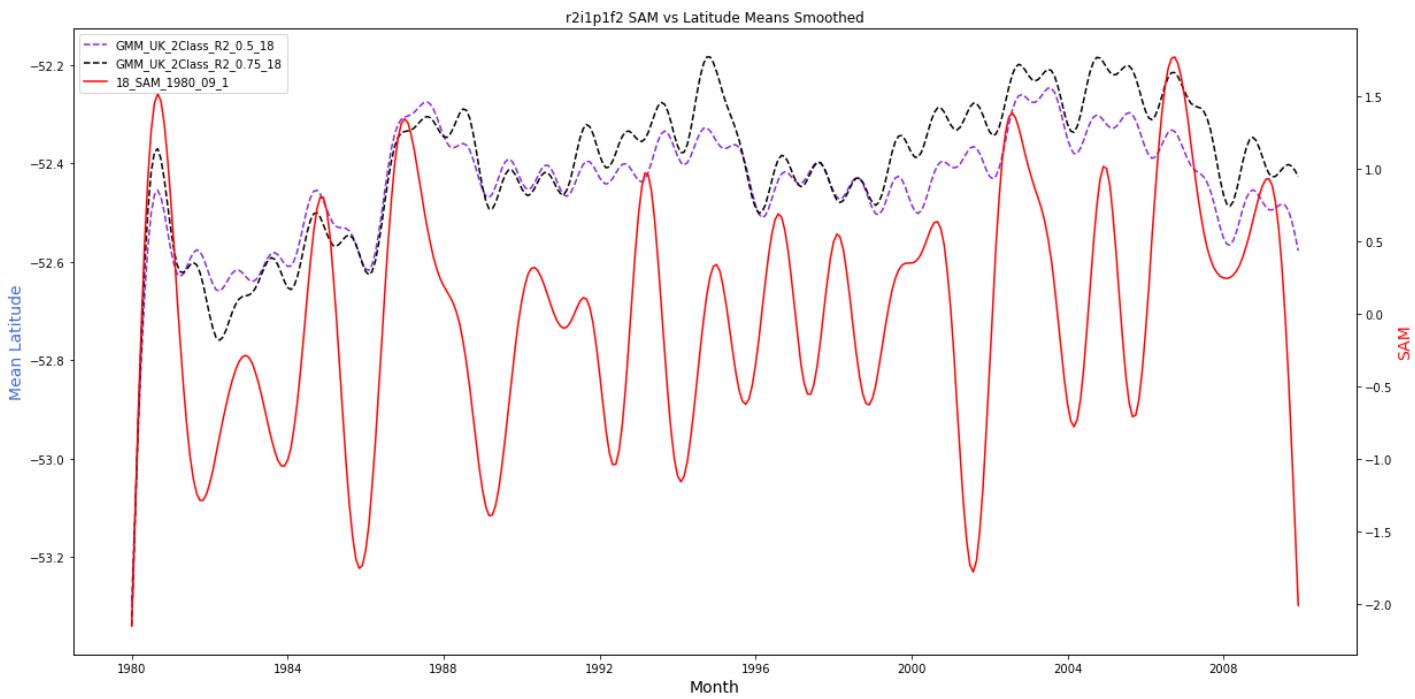
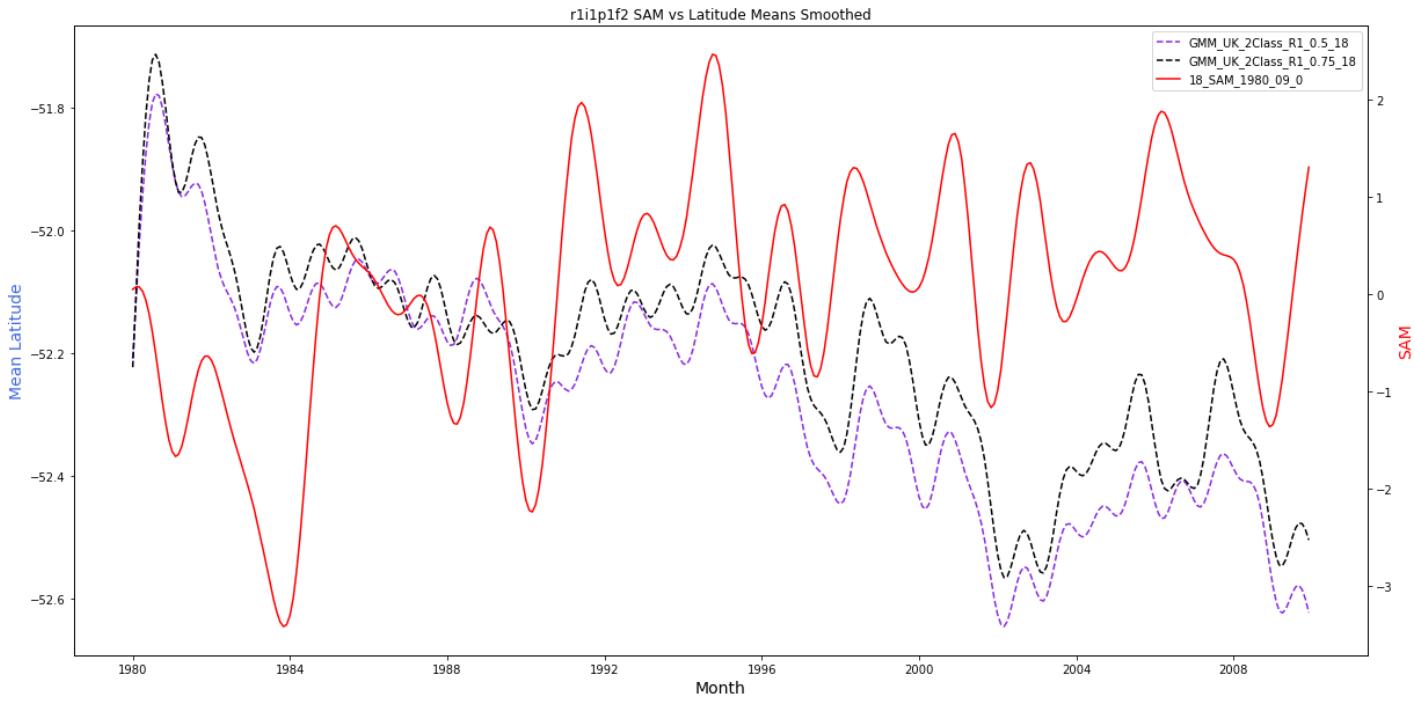
In [22]:

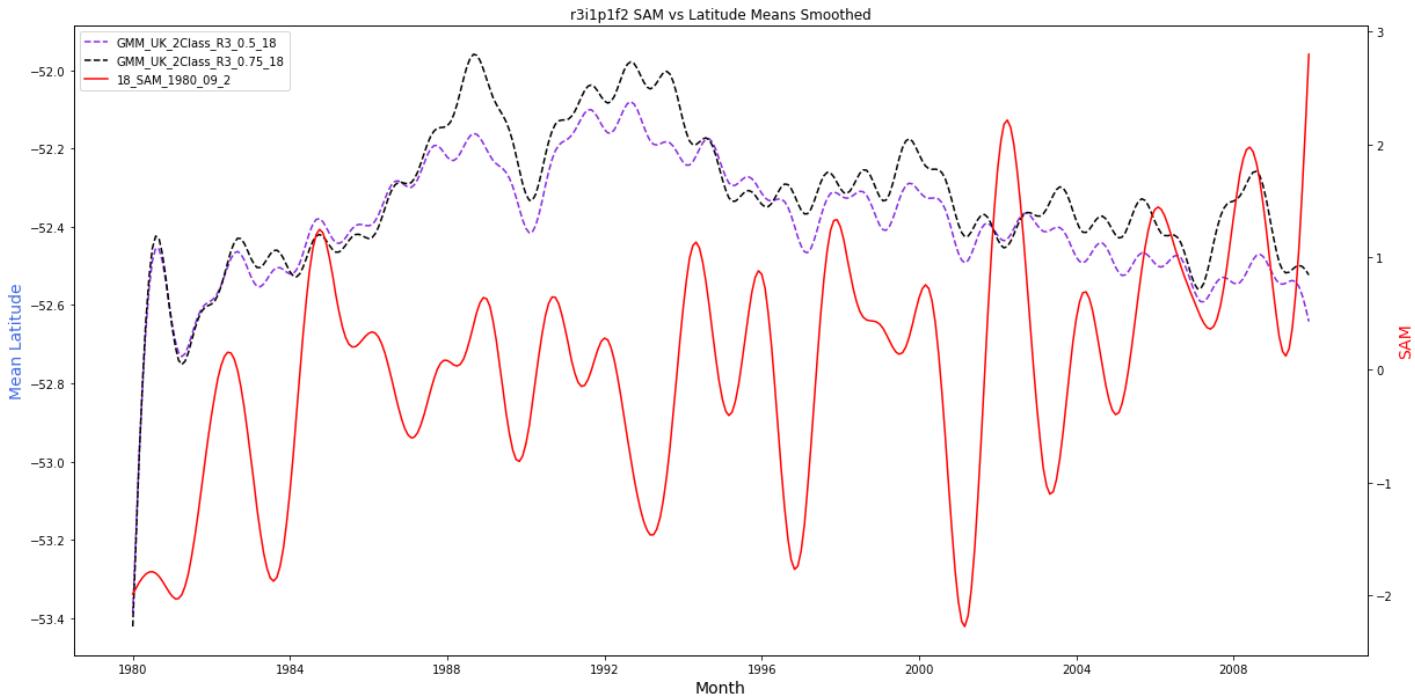
```
thresholdLook1 = 1
thresholdLook2 = 2
print("Smoothed SAM vs Latitude Means")
for i in range(fileCount):
    fig,ax = plt.subplots()
    fig.set_figheight(10)
    fig.set_figwidth(20)
    #ax.plot(timeAxis, LatMeanList[i][thresholdLook], label=str(GMModelNames[i])+"_"+str(i))
    ax.plot(timeAxis, smooth18Lat1980_09Arr[i][thresholdLook1], label=str(GMModelNames[i]))
    ax.plot(timeAxis, smooth18Lat1980_09Arr[i][thresholdLook2], label=str(GMModelNames[i]))
    ax.title.set_text(str(approvedIds[i]+" SAM vs Latitude Means Smoothed"))

    ax2 = ax.twinx()
    ax2.plot(SAM1980_09DF["time"], smooth18SAM1980_09Arr[i], label="18_SAM_1980_09_"+str(i))
    ax.set_xlabel("Month", fontsize=14)
    ax.set_ylabel("Mean Latitude", fontsize=14, color="RoyalBlue")
    ax2.set_ylabel("SAM", fontsize=14, color="Red")
    lines, labels = ax.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax2.legend(lines + lines2, labels + labels2)

plt.show()
```

Smoothed SAM vs Latitude Means





End of Notebook