class6

May 6, 2021

1 Doing the class 6 exp SRS in python

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import xarray as xr
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import netCDF4
import pandas as pd

import scipy.interpolate as interp
%matplotlib inline
```

[2]:
```
# Colormap selection
xr.set_options(cmap_divergent='bwr', cmap_sequential='turbo')
```

[2]:
```
<xarray.core.options.set_options at 0x7fd80c14d5b0>
```

[3]:
```
mfdataDIR1 = 'data/GPM/2009/3B-MO.MS.MRG.3IMERG.*.V06B.HDF5.SUB.nc4'
mfdataDIR2 = 'data/GPM/2019/3B-MO.MS.MRG.3IMERG.*.V06B.HDF5.SUB.nc4'

ds1 = xr.open_mfdataset(mfdataDIR1, parallel=True)
ds2 = xr.open_mfdataset(mfdataDIR2, parallel=True)
```

1.1 2009

[4]:
```
ds1
```

[4]:
```
<xarray.Dataset>
Dimensions:        (lat: 1800, lon: 3600, time: 12)
Coordinates:
  * time           (time) datetime64[ns] 2009-01-01 2009-02-01 … 2009-12-01
  * lon            (lon) float32 -179.9 -179.9 -179.8 … 179.8 179.9 179.9
  * lat            (lat) float32 -89.95 -89.85 -89.75 … 89.75 89.85 89.95
Data variables:
    precipitation  (time, lon, lat) float32 dask.array<chunksize=(1, 3600,
1800), meta=np.ndarray>
Attributes:
```

```
CDI:                                     Climate Data Interface version 1…
Conventions:                             CF-1.6
Original_Producer_Metadata_FileHeader:   DOI=10.5067/GPM/IMERG/3B-MONTH/06…
Original_Producer_Metadata_FileInfo:     DataFormatVersion=6a;\nTKCodeBuil…
Original_Producer_Metadata_GridHeader:   BinMethod=ARITHMETIC_MEAN;\nRegis…
InputPointer:                            3B-MO.MS.MRG.3IMERG.20090101-S000…
history_L34RS:                           'Created by L34RS v1.4.2 @ NASA G…
CDO:                                     Climate Data Operators version 1…
```

[5]:
```python
# make preciptation rate to preciptation

def convert_to_precipitaion(ds):
    temp = ds * 24
#     temp = temp.to_dataset()
    return temp
```

[6]:
```python
ds1 = convert_to_precipitaion(ds1)
```

[7]:
```python
# Transpose the data to get lat first and lon after -

ds1 = ds1.transpose("time", "lat", "lon")
```

[8]:
```python
ds1_ind = ds1.sel(lat=slice(7,36), lon=slice(67,98)).dropna("time")
```

[9]:
```python
# Wrap it into a simple function
def season_mean(ds, calendar='standard'):
    # Make a DataArray with the number of days in each month, size = len(time)
    month_length = ds.time.dt.days_in_month

    # Calculate the weights by grouping by 'time.season'
    weights = month_length.groupby('time.season') / month_length.groupby('time.
 ↪season').sum()

    # Test that the sum of the weights for each season is 1.0
    np.testing.assert_allclose(weights.groupby('time.season').sum().values, np.
 ↪ones(4))

    # Calculate the weighted average
    return (ds * weights).groupby('time.season').sum(dim='time')
```

[10]:
```python
# Get seasonal mean

ds1_ind_sm = season_mean(ds1_ind)

ds1_ind_sm
```

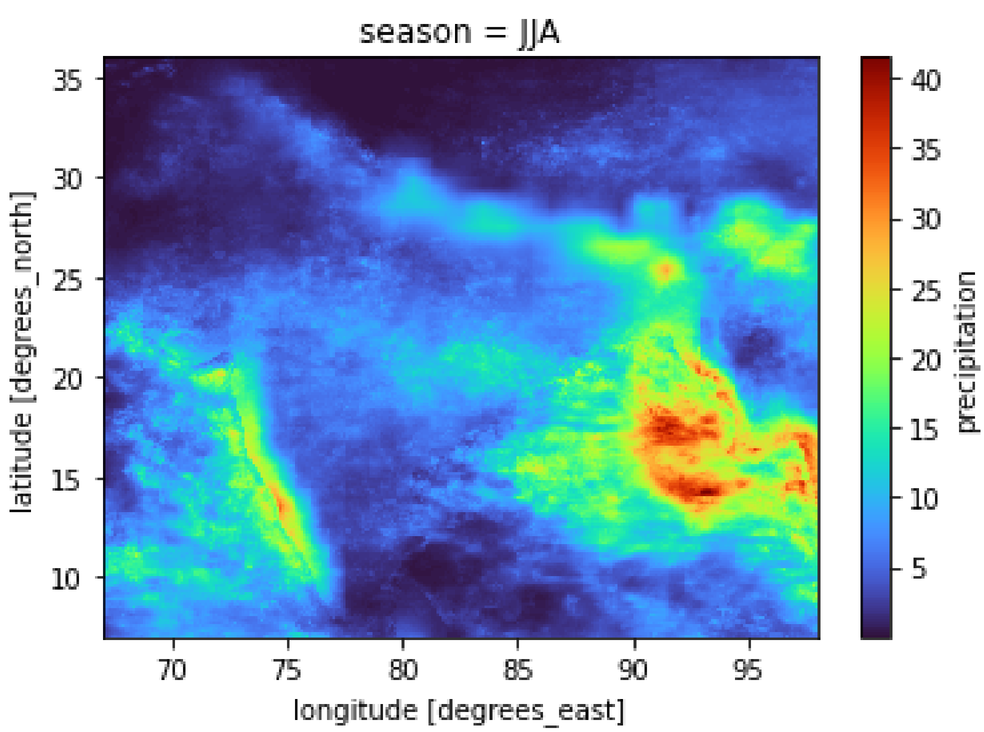
[10]: <xarray.Dataset>
```
Dimensions:        (lat: 290, lon: 310, season: 4)
Coordinates:
  * lon            (lon) float32 67.05 67.15 67.25 67.35 … 97.75 97.85 97.95
  * lat            (lat) float32 7.05 7.15 7.25 7.35 … 35.65 35.75 35.85 35.95
  * season         (season) object 'DJF' 'JJA' 'MAM' 'SON'
Data variables:
    precipitation  (season, lat, lon) float64 dask.array<chunksize=(1, 290,
310), meta=np.ndarray>
```

[11]:
```python
# Convert to dataarray

da1 = ds1_ind_sm.precipitation
da1
```

[11]: <xarray.DataArray 'precipitation' (season: 4, lat: 290, lon: 310)>
```
dask.array<concatenate, shape=(4, 290, 310), dtype=float64, chunksize=(1, 290,
310), chunktype=numpy.ndarray>
Coordinates:
  * lon      (lon) float32 67.05 67.15 67.25 67.35 … 97.65 97.75 97.85 97.95
  * lat      (lat) float32 7.05 7.15 7.25 7.35 7.45 … 35.65 35.75 35.85 35.95
  * season   (season) object 'DJF' 'JJA' 'MAM' 'SON'
```

[12]:
```python
da1.sel(season = 'JJA').plot()
```

[12]: <matplotlib.collections.QuadMesh at 0x7fd7b44ec370>



1.1.1 Attempting to mask the data

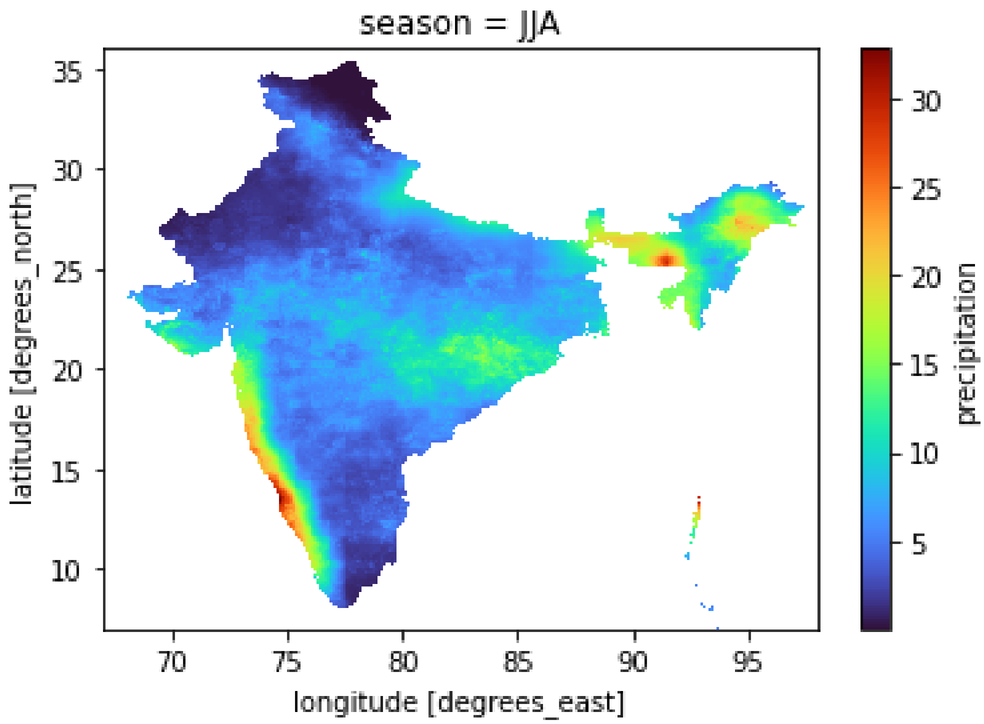
[13]:
```
import geopandas as gpd
from rasterio import features
from affine import Affine

def transform_from_latlon(lat, lon):
    """ input 1D array of lat / lon and output an Affine transformation
    """
    lat = np.asarray(lat)
    lon = np.asarray(lon)
    trans = Affine.translation(lon[0], lat[0])
    scale = Affine.scale(lon[1] - lon[0], lat[1] - lat[0])
    return trans * scale

def rasterize(shapes, coords, latitude='lat', longitude='lon',
              fill=np.nan, **kwargs):
    """Rasterize a list of (geometry, fill_value) tuples onto the given
    xray coordinates. This only works for 1d latitude and longitude
    arrays.

    usage:
    -----
    1. read shapefile to geopandas.GeoDataFrame
          `states = gpd.read_file(shp_dir+shp_file)`
    2. encode the different shapefiles that capture those lat-lons as different
        numbers i.e. 0.0, 1.0 ... and otherwise np.nan
          `shapes = (zip(states.geometry, range(len(states))))`
    3. Assign this to a new coord in your original xarray.DataArray
          `ds['states'] = rasterize(shapes, ds.coords, longitude='X',␣
↪latitude='Y')`

    arguments:
    ---------
    : **kwargs (dict): passed to `rasterio.rasterize` function

    attrs:
    -----
    :transform (affine.Affine): how to translate from latlon to ...?
    :raster (numpy.ndarray): use rasterio.features.rasterize fill the values
      outside the .shp file with np.nan
    :spatial_coords (dict): dictionary of {"X":xr.DataArray, "Y":xr.DataArray()}
      with "X", "Y" as keys, and xr.DataArray as values
```

```python
    returns:
    -------
    :(xr.DataArray): DataArray with `values` of nan for points outside shapefile
      and coords `Y` = latitude, 'X' = longitude.


    """
    transform = transform_from_latlon(coords['lat'], coords['lon'])
    out_shape = (len(coords['lat']), len(coords['lon']))
    raster = features.rasterize(shapes, out_shape=out_shape,
                                fill=fill, transform=transform,
                                dtype=float, **kwargs)
    spatial_coords = {latitude: coords['lat'], longitude: coords['lon']}
    return xr.DataArray(raster, coords=spatial_coords, dims=('lat', 'lon'))

def add_shape_coord_from_data_array(xr_da, shp_path, coord_name):
    """ Create a new coord for the xr_da indicating whether or not it
         is inside the shapefile

        Creates a new coord - "coord_name" which will have integer values
         used to subset xr_da for plotting / analysis/

        Usage:
        -----
        precip_da = add_shape_coord_from_data_array(precip_da, "awash.shp",␣
 ↪"awash")
        awash_da = precip_da.where(precip_da.awash==0, other=np.nan)
    """
    # 1. read in shapefile
    shp_gpd = gpd.read_file(shp_path)

    # 2. create a list of tuples (shapely.geometry, id)
    #    this allows for many different polygons within a .shp file (e.g.␣
 ↪States of US)
    shapes = [(shape, n) for n, shape in enumerate(shp_gpd.geometry)]

    # 3. create a new coord in the xr_da which will be set to the id in `shapes`
    xr_da[coord_name] = rasterize(shapes, xr_da.coords,
                               longitude='longitude', latitude='latitude')

    return xr_da
```

[14]:
```python
shp_dir = './shapefiles/'
```

[15]:
```python
da1_ind = add_shape_coord_from_data_array(da1, shp_dir, "awash")
awash_da1 = da1_ind.where(da1_ind.awash==0, other=np.nan)
awash_da1.sel(season="JJA").plot()
```

[15]: <matplotlib.collections.QuadMesh at 0x7fd782e37220>



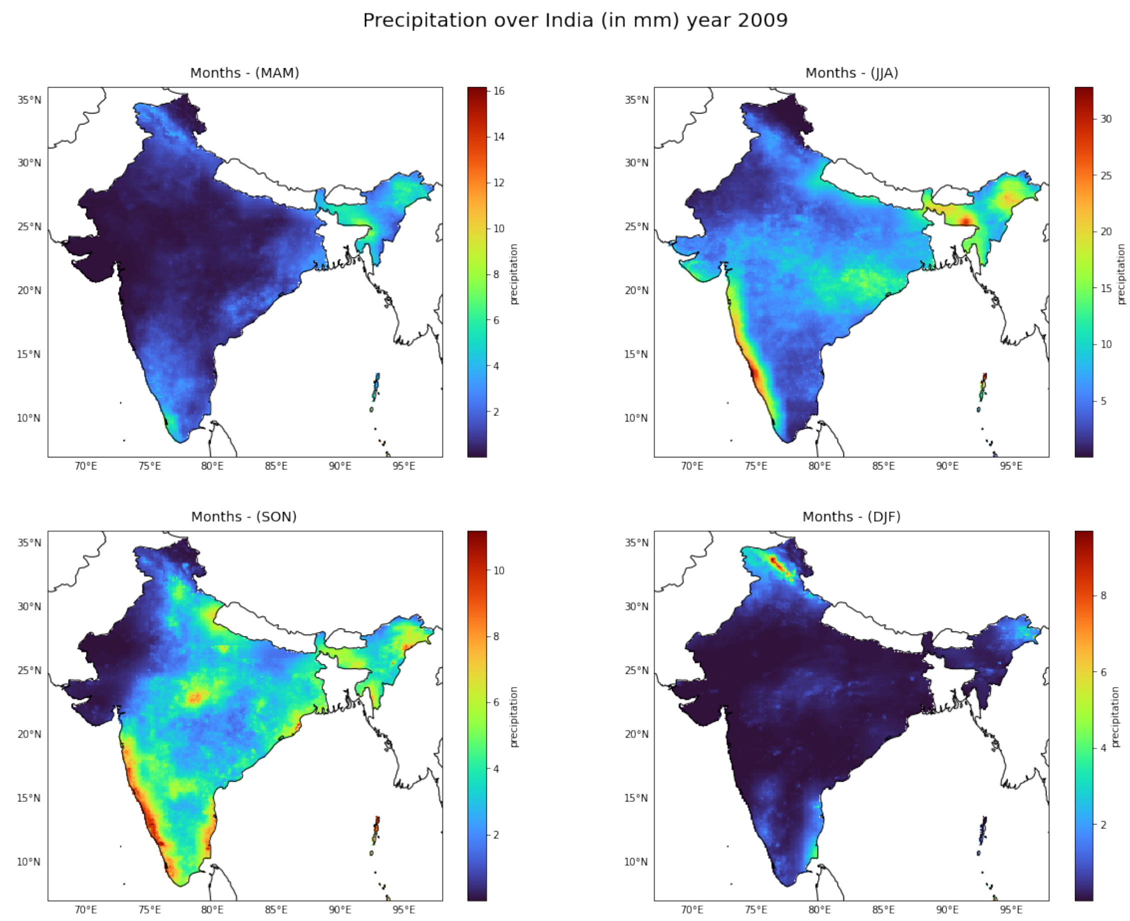
1.1.2 Take the different seasons and plot

[16]:
```
awash_da1
```

[16]:
```
<xarray.DataArray 'precipitation' (season: 4, lat: 290, lon: 310)>
dask.array<where, shape=(4, 290, 310), dtype=float64, chunksize=(1, 290, 310),
chunktype=numpy.ndarray>
Coordinates:
  * lon        (lon) float32 67.05 67.15 67.25 67.35 … 97.65 97.75 97.85 97.95
  * lat        (lat) float32 7.05 7.15 7.25 7.35 … 35.65 35.75 35.85 35.95
  * season     (season) object 'DJF' 'JJA' 'MAM' 'SON'
    latitude   (lat) float32 7.05 7.15 7.25 7.35 … 35.65 35.75 35.85 35.95
    longitude  (lon) float32 67.05 67.15 67.25 67.35 … 97.65 97.75 97.85 97.95
    awash      (lat, lon) float64 nan nan nan nan nan … nan nan nan nan nan
```

[17]:
```
# Plotting

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()

titles = ["MAM", "JJA", "SON", "DJF"]
```

```python
for i,season in enumerate(titles):

    ax = fig.add_subplot(2, 2, i+1, projection=ccrs.PlateCarree())
    ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
    awash_da1.sel(season=titles[i]).plot()
    gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
    gridliner.top_labels = False
    gridliner.bottom_labels = True
    gridliner.left_labels = True
    gridliner.right_labels = False
    gridliner.ylines = False  # you need False
    gridliner.xlines = False  # you need False
    ax.set_title("Months"+ " " + "-" + " " + "("+titles[i]+")", pad=10, fontsize=14)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS)

fig.suptitle('Precipitation over India (in mm) year 2009', fontsize=20, y=0.95)

plt.savefig('./images/GPM2009.png')
```



1.2 2019

[16]:
```python
ds2
```

[16]:
```
<xarray.Dataset>
Dimensions:          (lat: 1800, lon: 3600, time: 12)
Coordinates:
  * time             (time) datetime64[ns] 2019-01-01 2019-02-01 … 2019-12-01
  * lon              (lon) float32 -179.9 -179.9 -179.8 … 179.8 179.9 179.9
  * lat              (lat) float32 -89.95 -89.85 -89.75 … 89.75 89.85 89.95
Data variables:
    precipitation  (time, lon, lat) float32 dask.array<chunksize=(1, 3600,
1800), meta=np.ndarray>
Attributes:
    CDI:                                    Climate Data Interface version 1…
    Conventions:                            CF-1.6
    Original_Producer_Metadata_FileHeader:  DOI=10.5067/GPM/IMERG/3B-MONTH/06…
    Original_Producer_Metadata_FileInfo:    DataFormatVersion=6a;\nTKCodeBuil…
    Original_Producer_Metadata_GridHeader:  BinMethod=ARITHMETIC_MEAN;\nRegis…
    InputPointer:                           3B-MO.MS.MRG.3IMERG.20190101-S000…
    history_L34RS:                          'Created by L34RS v1.4.2 @ NASA G…
    CDO:                                    Climate Data Operators version 1…
```

[17]:
```python
# make preciptation rate to preciptation

ds2 = convert_to_precipitaion(ds2)
```

[18]:
```python
# Transpose the data to get lat first and lon after -

ds2 = ds2.transpose("time", "lat", "lon")
```

[19]:
```python
ds2_ind = ds2.sel(lat=slice(7,36), lon=slice(67,98)).dropna("time")
```

[20]:
```python
# Get seasonal mean

ds2_ind_sm = season_mean(ds2_ind)

ds2_ind_sm
```

[20]:
```
<xarray.Dataset>
Dimensions:          (lat: 290, lon: 310, season: 4)
Coordinates:
  * lon              (lon) float32 67.05 67.15 67.25 67.35 … 97.75 97.85 97.95
  * lat              (lat) float32 7.05 7.15 7.25 7.35 … 35.65 35.75 35.85 35.95
  * season           (season) object 'DJF' 'JJA' 'MAM' 'SON'
```

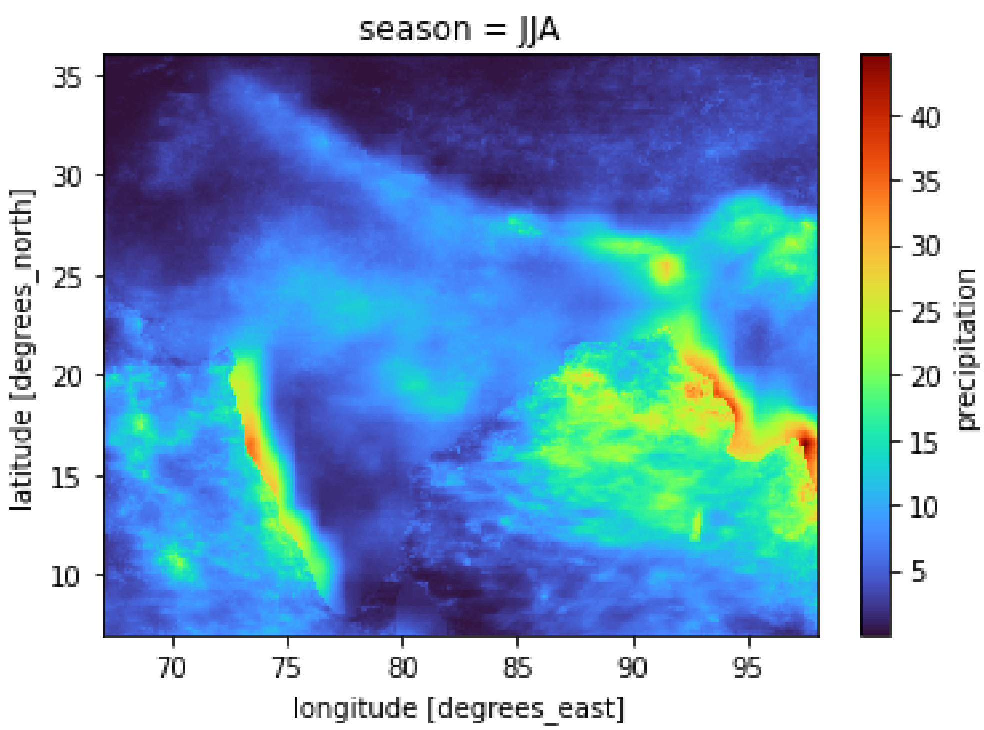
```
Data variables:
    precipitation  (season, lat, lon) float64 dask.array<chunksize=(1, 290,
310), meta=np.ndarray>
```

[21]:
```
# Convert to dataarray

da2 = ds2_ind_sm.precipitation
da2
```

[21]:
```
<xarray.DataArray 'precipitation' (season: 4, lat: 290, lon: 310)>
dask.array<concatenate, shape=(4, 290, 310), dtype=float64, chunksize=(1, 290,
310), chunktype=numpy.ndarray>
Coordinates:
  * lon      (lon) float32 67.05 67.15 67.25 67.35 … 97.65 97.75 97.85 97.95
  * lat      (lat) float32 7.05 7.15 7.25 7.35 7.45 … 35.65 35.75 35.85 35.95
  * season   (season) object 'DJF' 'JJA' 'MAM' 'SON'
```

[22]:
```
da2.sel(season = 'JJA').plot()
```

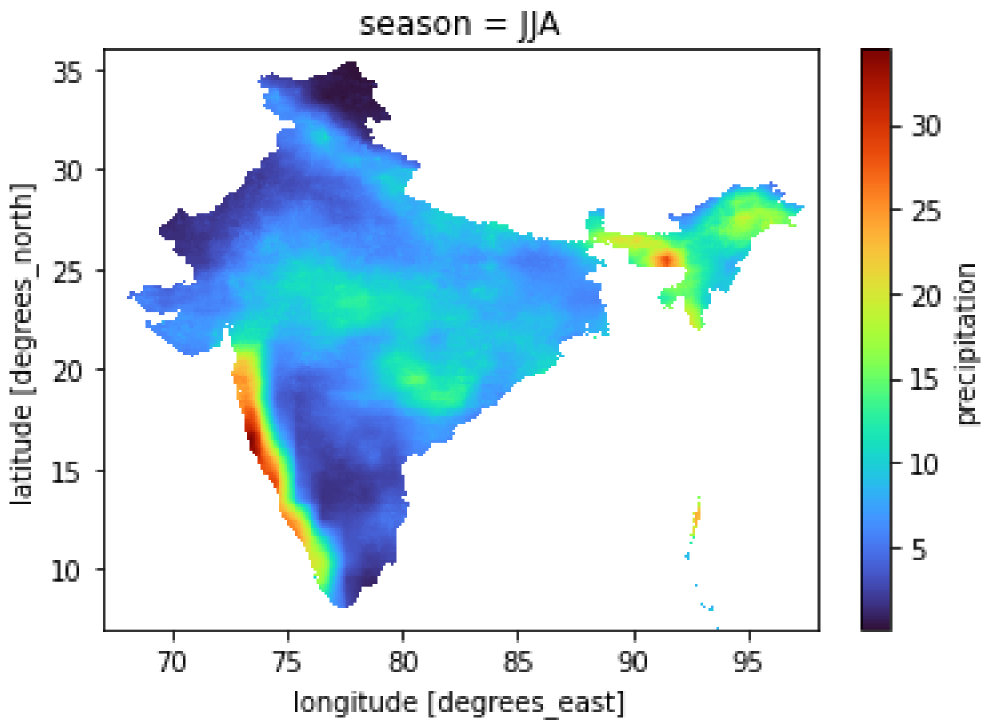
[22]:
```
<matplotlib.collections.QuadMesh at 0x7fd783da56a0>
```



[23]:
```
# Masking the data
```

```
da2_ind = add_shape_coord_from_data_array(da2, shp_dir, "awash")
awash_da2 = da2_ind.where(da2_ind.awash==0, other=np.nan)
awash_da2.sel(season="JJA").plot()
```

[23]: <matplotlib.collections.QuadMesh at 0x7fd78307e520>

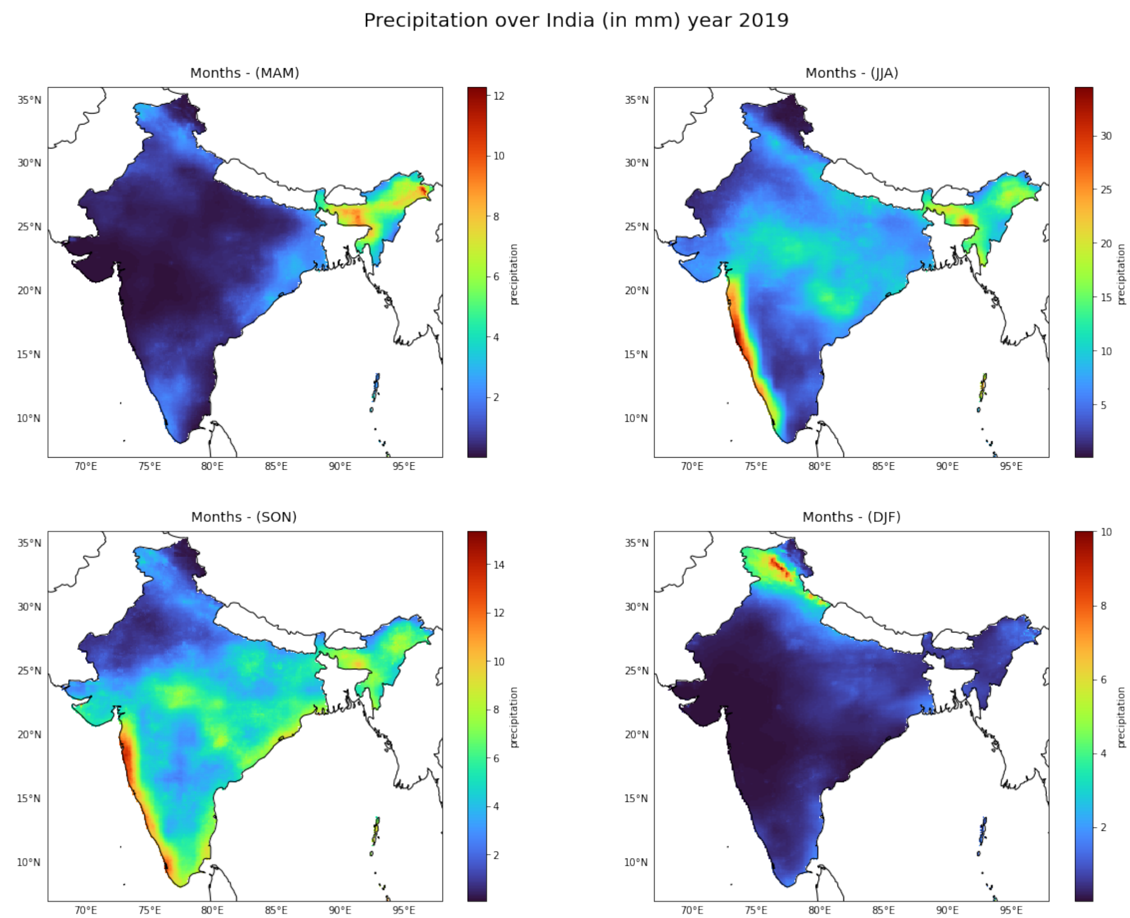


[26]:
```
awash_da2
```

[26]:
```
<xarray.DataArray 'precipitation' (season: 4, lat: 290, lon: 310)>
dask.array<where, shape=(4, 290, 310), dtype=float64, chunksize=(1, 290, 310),
chunktype=numpy.ndarray>
Coordinates:
  * lon        (lon) float32 67.05 67.15 67.25 67.35 … 97.65 97.75 97.85 97.95
  * lat        (lat) float32 7.05 7.15 7.25 7.35 … 35.65 35.75 35.85 35.95
  * season     (season) object 'DJF' 'JJA' 'MAM' 'SON'
    latitude   (lat) float32 7.05 7.15 7.25 7.35 … 35.65 35.75 35.85 35.95
    longitude  (lon) float32 67.05 67.15 67.25 67.35 … 97.65 97.75 97.85 97.95
    awash      (lat, lon) float64 nan nan nan nan nan … nan nan nan nan nan
```

[27]:
```
# Plotting

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()
```

```python
titles = ["MAM", "JJA", "SON", "DJF"]

for i,season in enumerate(titles):

    ax = fig.add_subplot(2, 2, i+1, projection=ccrs.PlateCarree())
    ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
    awash_da2.sel(season=titles[i]).plot()
    gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
    gridliner.top_labels = False
    gridliner.bottom_labels = True
    gridliner.left_labels = True
    gridliner.right_labels = False
    gridliner.ylines = False  # you need False
    gridliner.xlines = False  # you need False
    ax.set_title("Months"+ " " + "-" + " " + "("+titles[i]+")", pad=10, fontsize=14)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS)

fig.suptitle('Precipitation over India (in mm) year 2019', fontsize=20, y=0.95)
plt.savefig('./images/GPM2019.png')
```



1.3 Importing IMD data

[24]:
```
data3 = 'data/IMD/_Clim_Pred_LRF_New_RF25_IMD0p252009.nc'
data4 = 'data/IMD/_Clim_Pred_LRF_New_RF25_IMD0p252019.nc'

ds3 = xr.open_dataset(data3)
ds4 = xr.open_dataset(data4)
```

[25]:
```
ds3
```

[25]:
```
<xarray.Dataset>
Dimensions:    (LATITUDE: 129, LONGITUDE: 135, TIME: 365)
Coordinates:
  * LONGITUDE  (LONGITUDE) float64 66.5 66.75 67.0 67.25 … 99.5 99.75 100.0
  * LATITUDE   (LATITUDE) float64 6.5 6.75 7.0 7.25 … 37.75 38.0 38.25 38.5
  * TIME       (TIME) datetime64[ns] 2009-01-01 2009-01-02 … 2009-12-31
Data variables:
    RAINFALL   (TIME, LATITUDE, LONGITUDE) float32 …
Attributes:
    history:      FERRET V6.9   13-Jan-21
    Conventions:  CF-1.0
```

[26]:
```
# rename dimension names

ds3_ind = ds3.rename({"LONGITUDE":"lon", "LATITUDE":"lat","TIME":"time"})
```

[27]:
```
ds4_ind = ds4.rename({"LONGITUDE":"lon", "LATITUDE":"lat","TIME":"time"})
```

[28]:
```
# Getting seasonal mean for IMD data

ds3_ind_sm = season_mean(ds3_ind)

ds4_ind_sm = season_mean(ds4_ind)

ds3_ind_sm
```

[28]:
```
<xarray.Dataset>
Dimensions:   (lat: 129, lon: 135, season: 4)
Coordinates:
  * lon       (lon) float64 66.5 66.75 67.0 67.25 … 99.25 99.5 99.75 100.0
  * lat       (lat) float64 6.5 6.75 7.0 7.25 7.5 … 37.5 37.75 38.0 38.25 38.5
  * season    (season) object 'DJF' 'JJA' 'MAM' 'SON'
Data variables:
    RAINFALL  (season, lat, lon) float64 0.0 0.0 0.0 0.0 0.0 … 0.0 0.0 0.0 0.0
```

[29]:
```
gpm2009 = awash_da1
gpm2019 = awash_da2

imd2009 = ds3_ind_sm.RAINFALL
imd2019 = ds4_ind_sm.RAINFALL
```

1.3.1 Interpolating the IMD data like GPM

[30]:
```
# using interp_like

imd2009_interp = imd2009.interp_like(gpm2009)
imd2019_interp = imd2019.interp_like(gpm2019)
```

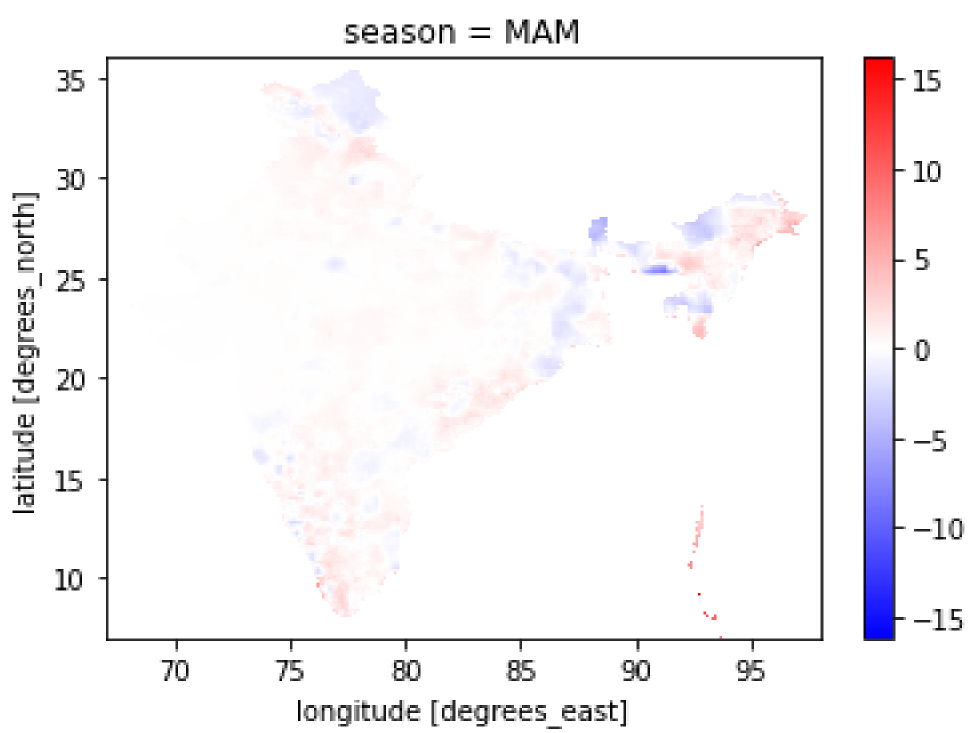
1.3.2 Calculation of performance metrics

Error

[31]:
```
# 2009 and 2019 GPM variation comparison to IMD

err2009 = gpm2009 - imd2009_interp
err2019 = gpm2019 - imd2019_interp
```

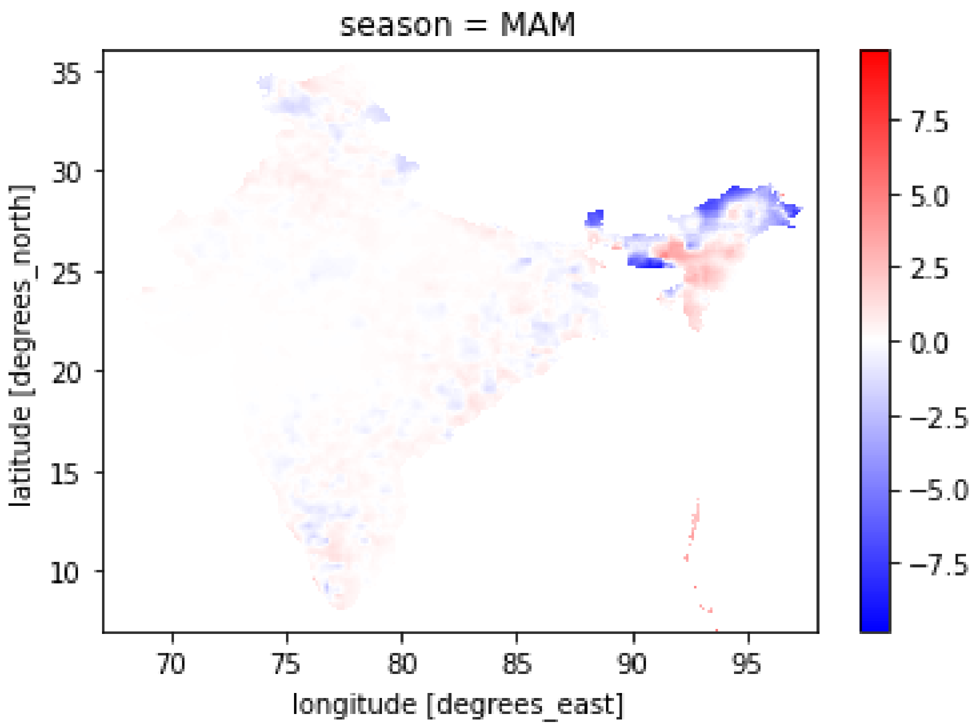
[32]:
```
err2009.sel(season = 'MAM').plot()
```

[32]: <matplotlib.collections.QuadMesh at 0x7fd780967c10>



[33]:
```
err2019.sel(season = 'MAM').plot()
```

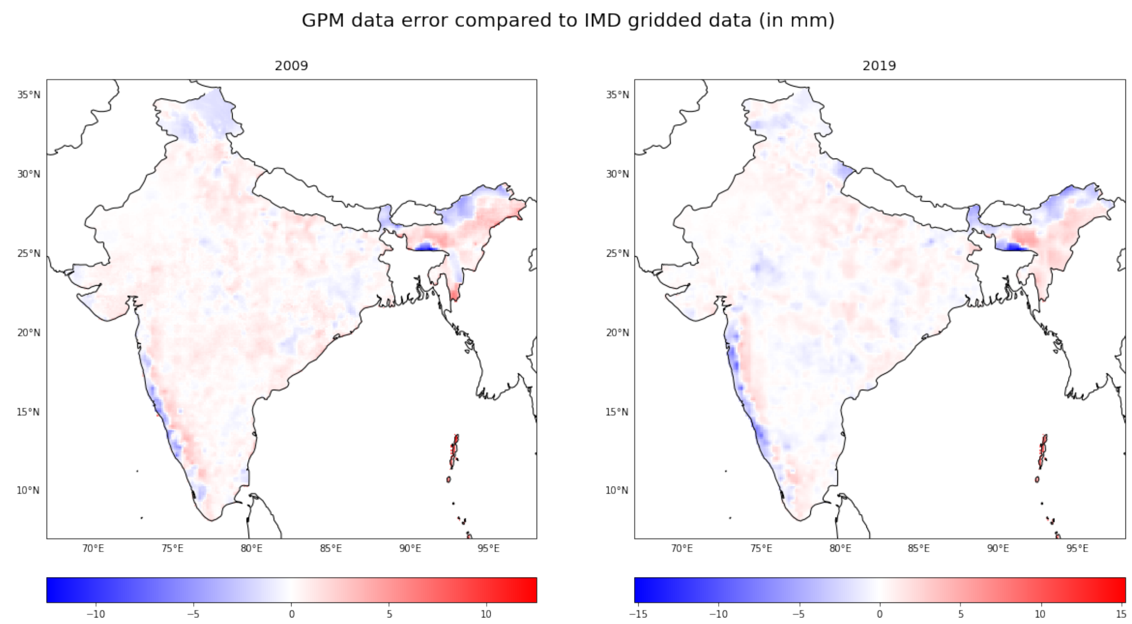
[33]: <matplotlib.collections.QuadMesh at 0x7fd7807bca00>



[35]:
```
# Plotting overall error for 2009 and 2019

# Plotting 2009 and 2019 RMSE

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()

ax = fig.add_subplot(1, 2, 1, projection=ccrs.PlateCarree())
ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
err2009.mean(dim='season').plot(cbar_kwargs = {"orientation":"horizontal",␣
 ↪"pad":0.05})
gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
gridliner.top_labels = False
gridliner.bottom_labels = True
gridliner.left_labels = True
gridliner.right_labels = False
```

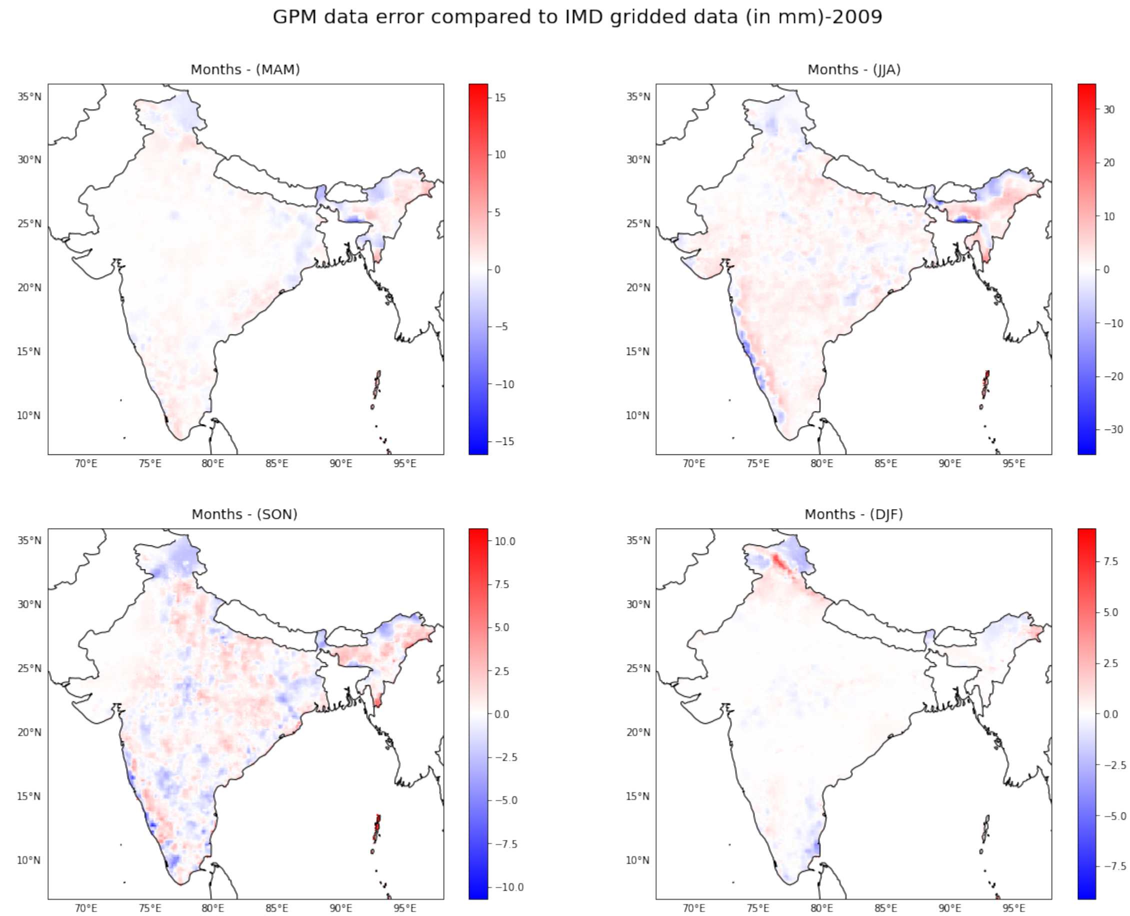
```python
gridliner.ylines = False  # you need False
gridliner.xlines = False  # you need False
ax.set_title("2009", pad=10, fontsize=14)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS)

ax = fig.add_subplot(1, 2, 2, projection=ccrs.PlateCarree())
ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
err2019.mean(dim='season').plot(cbar_kwargs = {"orientation":"horizontal",␣
↪"pad":0.05})
gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
gridliner.top_labels = False
gridliner.bottom_labels = True
gridliner.left_labels = True
gridliner.right_labels = False
gridliner.ylines = False  # you need False
gridliner.xlines = False  # you need False
ax.set_title("2019", pad=10, fontsize=14)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS)

fig.suptitle('GPM data error compared to IMD gridded data (in mm)',␣
↪fontsize=20, y=0.78)

plt.savefig('./images/err.png')
```

```
/home/aditya/.local/share/virtualenvs/atms_python-xEvIgfwt/lib/python3.9/site-
packages/dask/array/numpy_compat.py:39: RuntimeWarning: invalid value
encountered in true_divide
  x = np.divide(x1, x2, out)
/home/aditya/.local/share/virtualenvs/atms_python-xEvIgfwt/lib/python3.9/site-
packages/dask/array/numpy_compat.py:39: RuntimeWarning: invalid value
encountered in true_divide
  x = np.divide(x1, x2, out)
```



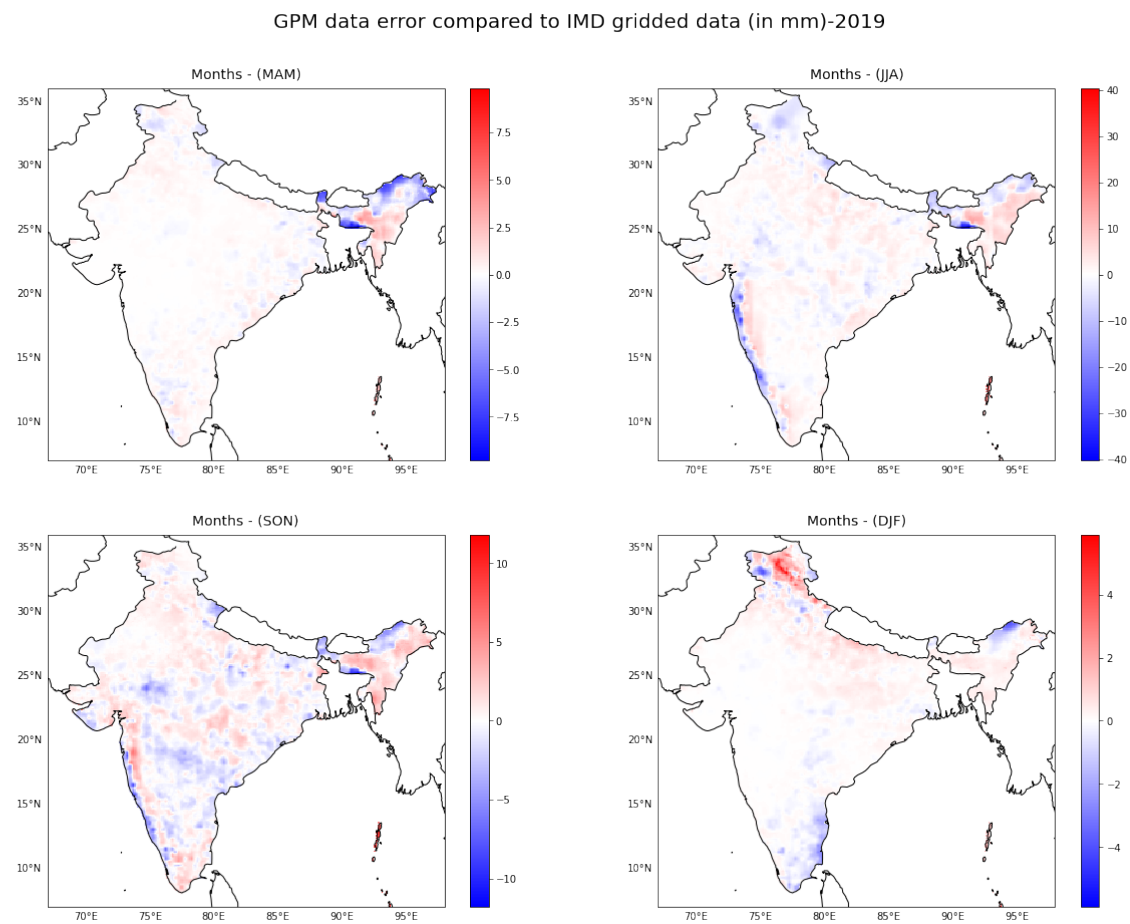
[38]:
```python
# Plotting 2009 seasonal error

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()

titles = ["MAM", "JJA", "SON", "DJF"]

for i,season in enumerate(titles):

    ax = fig.add_subplot(2, 2, i+1, projection=ccrs.PlateCarree())
    ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
    err2009.sel(season=titles[i]).plot()
    gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
    gridliner.top_labels = False
    gridliner.bottom_labels = True
    gridliner.left_labels = True
    gridliner.right_labels = False
    gridliner.ylines = False  # you need False
    gridliner.xlines = False  # you need False
    ax.set_title("Months"+ " " + "-" + " " + "("+titles[i]+")", pad=10, fontsize=14)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS)

fig.suptitle('GPM data error compared to IMD gridded data (in mm)-2009', fontsize=20, y=0.95)
```

```
plt.savefig('./images/err2009.png')
```



[39]:
```
# Plotting 2019 seasonal error

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()

titles = ["MAM", "JJA", "SON", "DJF"]

for i,season in enumerate(titles):

    ax = fig.add_subplot(2, 2, i+1, projection=ccrs.PlateCarree())
    ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
    err2019.sel(season=titles[i]).plot()
    gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
    gridliner.top_labels = False
    gridliner.bottom_labels = True
```

```
    gridliner.left_labels = True
    gridliner.right_labels = False
    gridliner.ylines = False  # you need False
    gridliner.xlines = False  # you need False
    ax.set_title("Months"+ " " + "-" + " " + "("+titles[i]+")", pad=10,␣
↪fontsize=14)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS)

fig.suptitle('GPM data error compared to IMD gridded data (in mm)-2019',␣
↪fontsize=20, y=0.95)

plt.savefig('./images/err2019.png')
```



RMSE

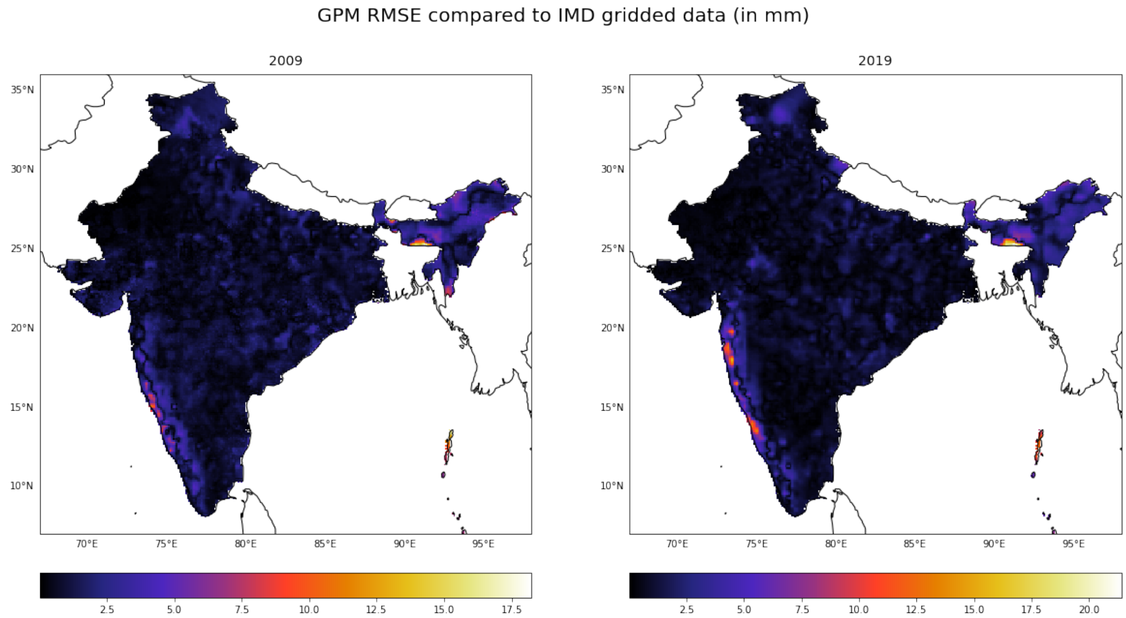
[64]:
```
# Resetting Colormap selection for RMSE
```

```
xr.set_options(cmap_divergent='bwr', cmap_sequential='CMRmap') # divergent␣
↪doesn't matter here
```

[64]: <xarray.core.options.set_options at 0x7f846d800ac0>

[40]:
```
rmse2009 = np.sqrt((err2009 * err2009).mean(dim = 'season'))
rmse2019 = np.sqrt((err2019 * err2019).mean(dim = 'season'))
```

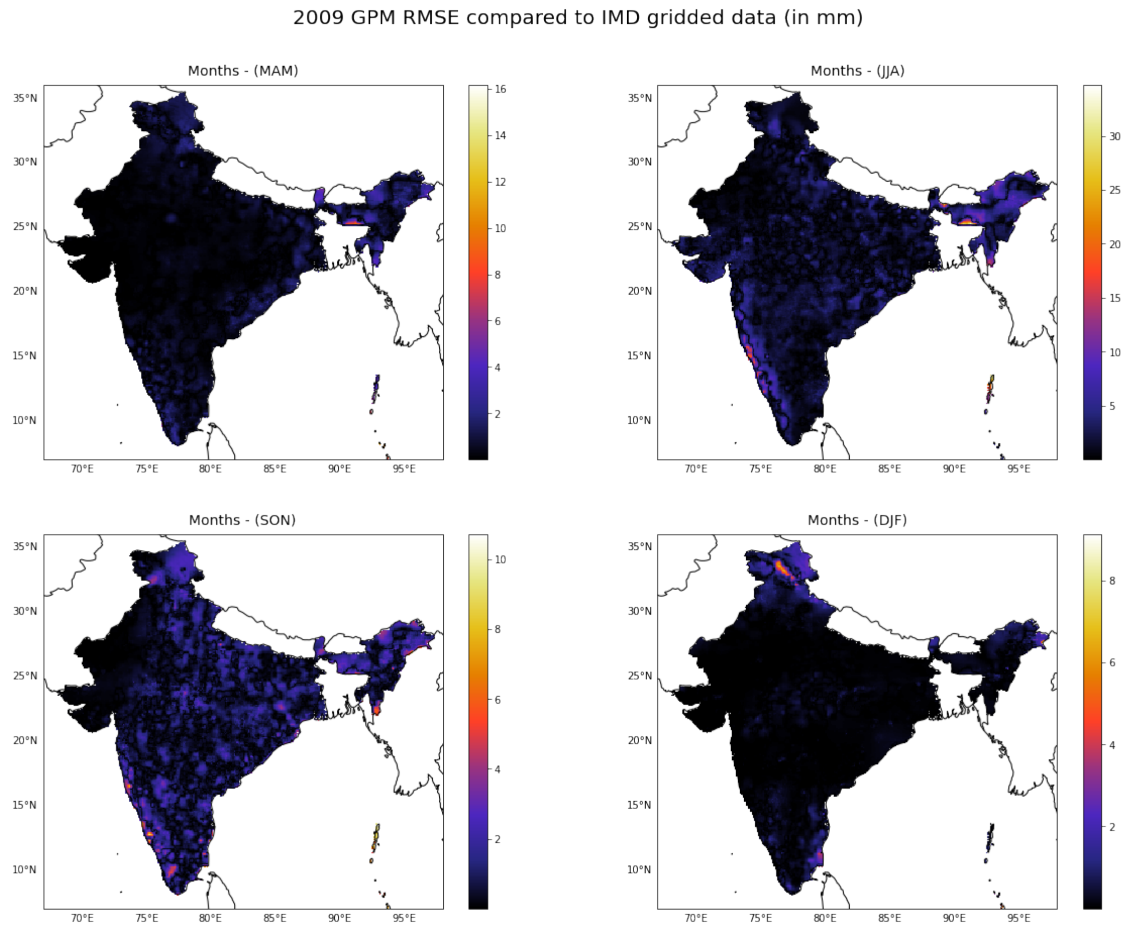
[65]:
```
# Plotting 2009 and 2019 RMSE

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()

ax = fig.add_subplot(1, 2, 1, projection=ccrs.PlateCarree())
ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
rmse2009.plot(cbar_kwargs = {"orientation":"horizontal", "pad":0.05})
gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
gridliner.top_labels = False
gridliner.bottom_labels = True
gridliner.left_labels = True
gridliner.right_labels = False
gridliner.ylines = False  # you need False
gridliner.xlines = False  # you need False
ax.set_title("2009", pad=10, fontsize=14)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS)

ax = fig.add_subplot(1, 2, 2, projection=ccrs.PlateCarree())
ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
rmse2019.plot(cbar_kwargs = {"orientation":"horizontal", "pad":0.05})
gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
gridliner.top_labels = False
gridliner.bottom_labels = True
gridliner.left_labels = True
gridliner.right_labels = False
gridliner.ylines = False  # you need False
gridliner.xlines = False  # you need False
ax.set_title("2019", pad=10, fontsize=14)
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS)

fig.suptitle('GPM RMSE compared to IMD gridded data (in mm)', fontsize=20, y=0.
↪78)

plt.savefig('./images/rmse.png')
```



[66]:
```
# define a function to calculate rmse error for given dataarray error value

def rmse_calc(da_err, season):
    """
    The RMSE calc function calculates the rmse from given input error value
    and also takes a season string as input for selecting the seasonal
    mean whose rmse needs to be calculated
    """
    months = ['MAM','JJA','SON','DJF']
    if season == 'MAM':
        rmse = np.sqrt((da_err * da_err).sel(season = months[0]))
    elif season == 'JJA':
        rmse = np.sqrt((da_err * da_err).sel(season = months[1]))
    elif season == 'SON':
        rmse = np.sqrt((da_err * da_err).sel(season = months[2]))
    elif season == 'DJF':
        rmse = np.sqrt((da_err * da_err).sel(season = months[3]))
    else:
        print("ERROR : Please enter a correct season value")

    return rmse
```

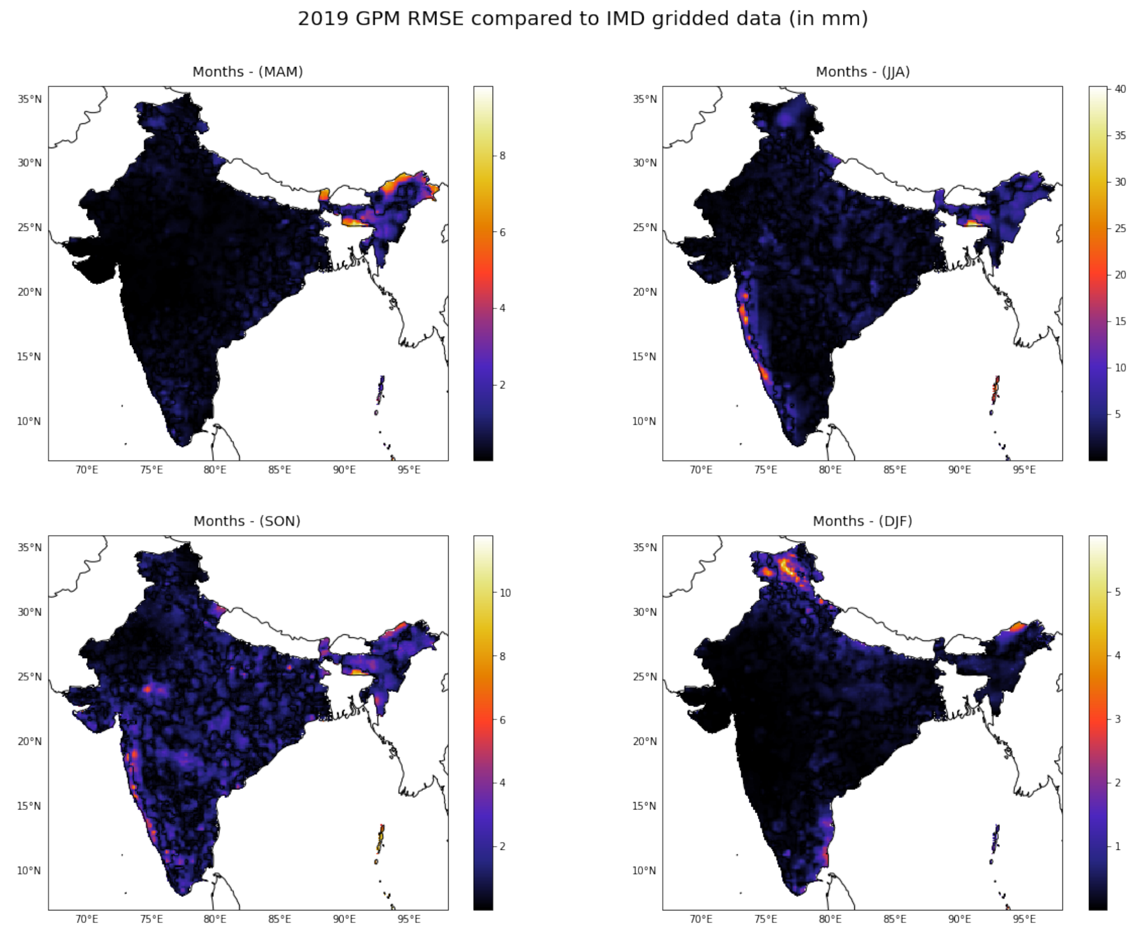
[67]:
```
# Plotting rmse error for seasonal means for 2009

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()
titles = ["MAM", "JJA", "SON", "DJF"]
```

```
for i,season in enumerate(titles):

    ax = fig.add_subplot(2, 2, i+1, projection=ccrs.PlateCarree())
    ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
    rmse_calc(err2009, titles[i]).plot()
    gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
    gridliner.top_labels = False
    gridliner.bottom_labels = True
    gridliner.left_labels = True
    gridliner.right_labels = False
    gridliner.ylines = False  # you need False
    gridliner.xlines = False  # you need False
    ax.set_title("Months"+ " " + "-" + " " + "("+titles[i]+")", pad=10,␣
↪fontsize=14)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS)

fig.suptitle('2009 GPM RMSE compared to IMD gridded data (in mm)', fontsize=20,␣
↪y=0.95)
plt.savefig('./images/rmse2009.png')
```




```
[68]: # Plotting rmse error for seasonal means for 2019

fig = plt.figure(figsize=(20, 15))
fig.tight_layout()
titles = ["MAM", "JJA", "SON", "DJF"]

for i,season in enumerate(titles):

    ax = fig.add_subplot(2, 2, i+1, projection=ccrs.PlateCarree())
    ax.set_extent([67, 98, 7, 36], crs=ccrs.PlateCarree())
    rmse_calc(err2019, titles[i]).plot()
    gridliner = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True)
    gridliner.top_labels = False
    gridliner.bottom_labels = True
    gridliner.left_labels = True
    gridliner.right_labels = False
    gridliner.ylines = False  # you need False
    gridliner.xlines = False  # you need False
```

```
    ax.set_title("Months"+ " " + "-" + " " + "("+titles[i]+")", pad=10,␣
↪fontsize=14)
    ax.add_feature(cfeature.COASTLINE)
    ax.add_feature(cfeature.BORDERS)

fig.suptitle('2019 GPM RMSE compared to IMD gridded data (in mm)', fontsize=20,␣
↪y=0.95)
plt.savefig('./images/rmse2019.png')
```



1.3.3 Time series plot of both IMD and GPM data

[141]:
```
# Function to convert the IMD daily data to one month data
# we do the same with GPM for consistency

def daily_to_monthly(da):

    temp = da.groupby('time.month').mean(dim='time')
    times = pd.date_range(start = "2009-01-15", end = "2010-01-15",freq='M')
```

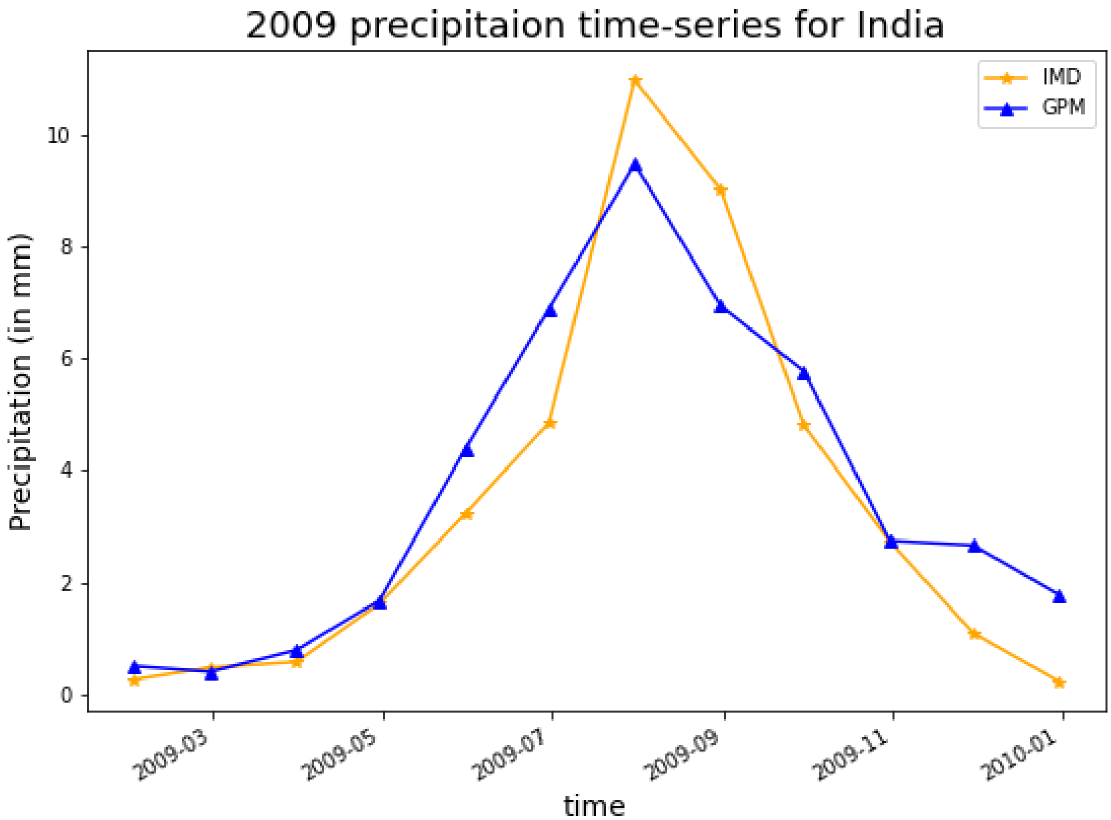
```
    output = xr.DataArray(
        temp,
        coords={
            "time": times,
            "lon": temp.lon,
            "lat": temp.lat
        },
        dims=["time", "lat", "lon"],
    )

    return output
```

[142]:

```
# get the time series dataarrays using the above function. Here "ts" is␣
↪timeseries

gpm2009_ts = daily_to_monthly(ds1_ind.precipitation)
imd2009_ts = daily_to_monthly(ds3_ind.RAINFALL)

gpm2019_ts = daily_to_monthly(ds2_ind.precipitation)
imd2019_ts = daily_to_monthly(ds4_ind.RAINFALL)
```

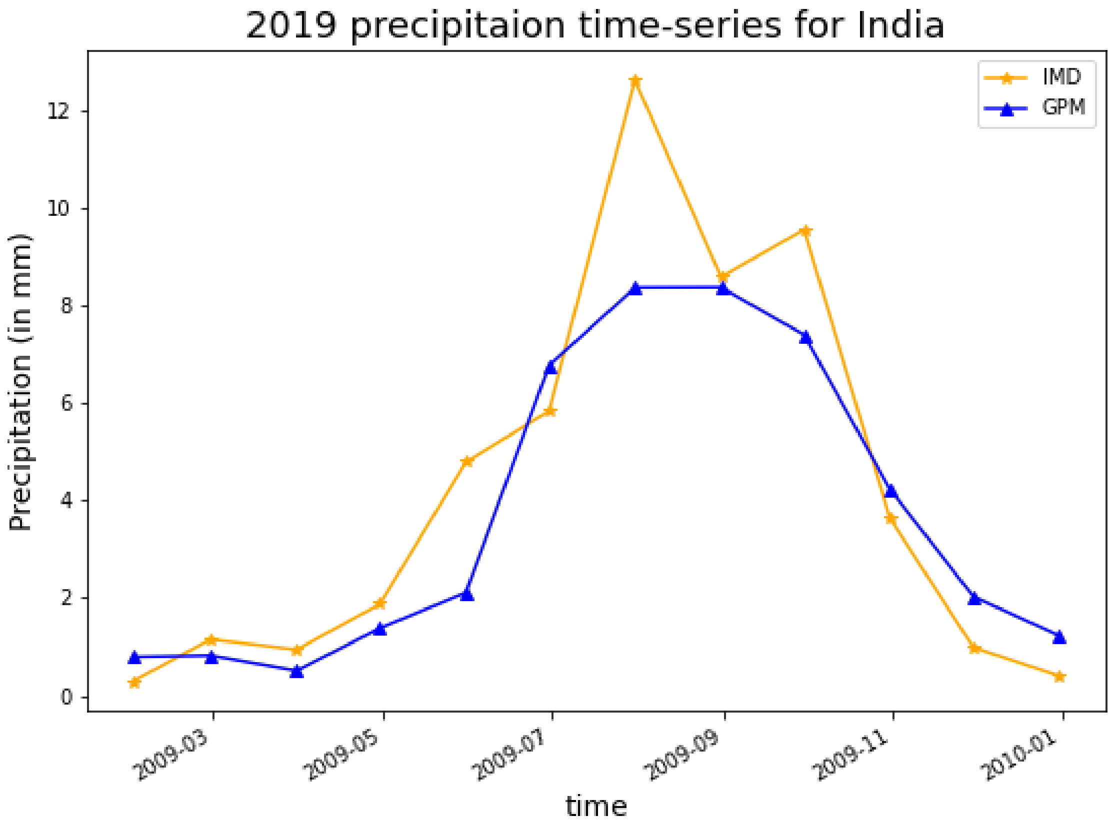
[215]:

```
# Time series 2009

fig= plt.figure(figsize=(9,6))
imd2009_ts.mean(dim='lat').mean(dim='lon').plot.line("-*",color='orange', label␣
↪= 'IMD')
gpm2009_ts.mean(dim='lat').mean(dim='lon').plot.line("b-^", label = 'GPM')

plt.legend()
plt.title('2009 precipitaion time-series for India', fontdict={"size":18})
plt.ylabel('Precipitation (in mm)', fontdict={"size":14})
plt.xlabel('time',fontdict={"size":14})

plt.savefig('./images/time_series2009.png')
```



[216]:

```
# Time series 2019

fig = plt.figure(figsize=(9,6))
imd2019_ts.mean(dim='lat').mean(dim='lon').plot.line("-*",color='orange', label␣
↪= 'IMD')
gpm2019_ts.mean(dim='lat').mean(dim='lon').plot.line("b-^", label = 'GPM')

plt.legend()
plt.title('2019 precipitaion time-series for India', fontdict={"size":18})
plt.ylabel('Precipitation (in mm)', fontdict={"size":14})
plt.xlabel('time',fontdict={"size":14})

plt.savefig('./images/time_series2019.png')
```


2019 precipitaion time-series for India
IMD
GPM
Precipitation (in mm)
0
2
4
6
8
10
12
2009-03
2009-05
2009-07
2009-09
2009-11
2010-01
time

```
[ ]:
```