mmDiagnosis Documentation

Release 1a

Arulalan.T, Dr.Krishna AchutaRao, Dilipkumar.R

April 13, 2015

Contents:

CHAPTER

ONE

GETTING STARTED

1.1 Installing uv-cdat in system

To install uv-cdat in your system, please follow the following links.

http://uv-cdat.llnl.gov/wiki/CDATCMakeBuild

http://uv-cdat.llnl.gov/wiki/CDATBuild

1.2 Installing the Diagnosisutils

Have to write it

1.3 Installing the Diagnosis

Have to write it

1.4 Setup and Configuration

Have to write it

CHAPTER

TWO

DOCUMENTATION OF DIAGNOSISUTILS SOURCE CODE

The diagnosisutils package contains the Data Access Utils, Time Axis Utils, and Plot Utils modules.

2.1 Data Access Utils

This data access utils uses the Time Axis Utils and **'Days Utils'_** .

The xml_data_access module help us to access the all the grib files through single object.

Basically all the grib files are pointed into xml dom by cdscan command.

Then we can the xml files through uv-cdat.

Right now we are generating 8 xml files to access analysis grib files and 7 forecasts grib files.

In the xml_data_access module, we are finding which xml needs to access depends upon the user inputs (Type, hour) in the functions of this module.

And once one xml object has initiated then through out that program execute session, it will remains exists and use when ever user needs it.

2.1.1 xml_data_access

class `xml_data_access.`**`GribXmlAccess`**(*XmlDir*)

xml access methods

`closeXmlObjs`()

`closeXmlObjs()`: close all the opened xml file objects by cdms2. If we called this method, it will check all the 8 xml objects are either opened or not. If that is opened by cdms2 means, it will close that file object properly. We must call this method for the safety purpose.

..note:: If we called this method, at the end of the program then it should be optimized one. If this method called at any inter mediate level means, then again it need to create the xml object.

Written By: Arulalan.T

Date : 10.09.2011

`findPartners`(*Type*, *date*, *hour=None*, *returnType='c'*)

`findPartners()`: To find the partners of the any particular day anl or any particular day and hour of the fcst. Each fcst file(day) has its truth anl file(day). i.e. today 24 hour fcst file's partner is tomorrow's truth anl file. today 48 hour fcst file's partner is the day after tomorrow's truth anl file. Keep going on the fcst vc anl files.

Same concept for anl files partner but in reverse concept. Today's truth anl file's partners are yesterdays' 24 hour fcst file, day before yesterday's 48 hour fcst file and keep going backward ...

This what we are calling as the partners of anl and fcst files. For present fcst hours partner is future anl file and for present anl partners are the past fcst hours files.

Condition :

if 'f' as passed then hour is mandatory one else 'a' as passed then hour is optional one. returnType either 'c' or 's'

Inputs :

Type = 'f' or 'a' or 'o' i.e fcst or anl or obs file date must be cdtime.comptime object or its string formate hour is like 24 multiples in case availability of the fcst files

Outputs :

If 'f' has passed this method returns a corresponding partner of the anlysis date in cdtime.comptime object If 'a' or 'o' has passed this method returns a dictionary. It contains the availability of the fcst hours as key and its corresponding fcst date in cdtime.comptime object as value of the dict.

we can get the return date as yyyymmdd string formate by passing returnType = 's'

Usage :

example 1 :

```
>>> findPartners('f','2010-5-25',24)
    2010-5-26 0:0:0.0
```

Note: The passed date in comptime in string type.

```
>>> findPartners('f',cdtime.comptime(2010,5,25),24)
    2010-5-26 0:0:0.0
```

Note: The passed date in comptime object itself.

```
>>> findPartners('a','2010-5-26')
    {24: 2010-5-25 0:0:0.0}
```

Note: Returns dictionary which contains key as hour and its corresponding date

example 2 :

```
>>> findPartners('f','2010-5-25',72)
    2010-5-28 0:0:0.0
```

```
>>> findPartners('a','2010-6-1', returnType ='s')
     {24: '20100531',
      48: '20100530',
      72: '20100529',
      96: '20100528',
     120: '20100527',
     144: '20100526',
     168: '20100525'}
```

Note: Depends upon the availability of the fcst and anl files, it should return partner date

---

example 3 :

```
>>> findPartners('a','20100601',144)
    2010-5-26 0:0:0.0
```

See also:

If not available for the passed hour means it should return None

Written by : Arulalan.T

Date : 03.04.2011

`getData`(*var*, *Type*, *date*, *hour=None*, *level='all'*, ***latlonregion*)

`getData()`: It can extract either the data of a single date or range of dates. It depends up on the input of the date argument. Finally it should return MV2 variable.

Condition : date is either tuple or string. level is optional. level takes default 'all'. if level passed, it must be belongs to the data variable hour is must when Type arg should be 'f' (fcst) to choose xml object. Pass either (lat,lon) or region.

Inputs :

Type - either 'a'[analysis] or 'f'[forecast] or 'o'[observation] var,level must be belongs to the data file key word arg lat,lon or region should be passed date formate must one of the followings

date formate 1: date = (startdate, enddate) here startdate and enddate must be like cdtime.comptime formate.

date formate 2: date = (startdate)

date formate 3: date = 'startdate' or date = 'date'

eg for date input : date = ('2010-5-1','2010-6-30') date = ('2010-5-30') date = '2010-5-30'

Outputs :

If user passed single date in the date argument, then it should return the data of that particular date as single MV2 variable.

If user passed start and enddate in the date argument, then it should return the data for the range of dates as single MV2 variable with time axis.

Written by: Arulalan.T

Date: 10.05.2011

`getDataPartners`(*var*, *Type*, *date*, *hour=None*, *level='all'*, *orginData=0*, *datePriority='o'*, ***latlonregion*)

`getDataPartners()`: It can extract either the orginDate with its partnersData or it can extract only the partnersData without its orginData for a single date or range of dates.

It depends up on the input of the orginData, datePriority,date args. Finally it should return partnersData and/or orginData as MV2 variable

Condition :

date is either tuple or string. level is optional. level takes default 'all'. if level passed, it must be belongs to the data variable hour is must when Type arg should be 'f' (fcst) to select xml object. hour is must when range of date passed, even thogut Type arg should be 'a'(anl) or 'o'(obs),to choose one fcst xml object along with hour. Pass either (lat,lon) or regionself.

Inputs:

Type - either 'a'[analysis] or 'f'[forecast] or 'o'[observation] var,level must be belongs to the data file orginData - either 0 or 1. 0 means it shouldnot return the

orginData as single MV2 var. 1 means it should return both the orginData and its partnersData as two seperate MV2 vars.

datePriority - either 'o' or 'p'. 'o' means passed date is with

respect to orginData. According to this orginData's date, it should return its partnersData.

'p' means passed date is with respect to partnersData. According to this partnersData's date, it should return its orginData.

key word arg lat,lon or region should be passed

date formate 1: date = (startdate,enddate) here startdate and enddate must be like cdtime.comptime formate.

date formate 2: date = (startdate)

date formate 3: date = 'startdate' or date = 'date'

eg for date input :

date = ('2010-5-1','2010-6-30') date = ('2010-5-30') date = '2010-5-30'

Outputs:

If user passed single date in the date argument, then it should return the data of that particular date (both orginData & partnersData) as a single MV2 variable.

If user passed start and enddate in the date argument, then it should return the data (both orginData & partnersData) for the range of dates as a single MV2 variable with time axis.

Usage:

Note: if 'a'(anl) file is orginData means 'f'(fcst) files are its partnersData and vice versa.

example1:

```
>>> a,b = getDataPartners(var = 'U component of wind',Type = 'a',
        date = '2010-6-5',hour = None,level = 'all',orginData = 1,
        datePriority = 'o', lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e. anl. its timeAxis date is '2010-6-5'. b is partnersData. i.e. fcst. its 24 hour fcst date w.r.t orginData is '2010-6-4'. 48 hour is '2010-6-3'.

Depends upon the availability of date of fcst files,it should return the data. In NCMRWF2010 model, it should return maximum of 7 days fcst.

If we will specify any hour in the same eg, that should return only that hour fcst file data instead of returning all the available fcst hours data.

example2:

```
>>> a,b = getDataPartners(var = 'U component of wind',Type = 'f',
          date = '2010-6-5',hour = 24,level = 'all',orginData = 1,
          datePriority = 'o', lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e.fcst 24 hour.its timeAxis date is '2010-6-5'. b is partnersData. i.e. anl. its anl date w.r.t orginData is '2010-6-6'.

example3:

```
>>> b = getDataPartners(var = 'U component of wind',Type = 'f',
         date = '2010-6-5',hour = 24,level = 'all', orginData = 0,
         datePriority = 'o', lat=(-90,90),lon=(0,359.5))
```

b is partnersData. i.e. anl. its anl date w.r.t orginData is '2010-6-6'. No orginData. Because we passed orginData as 0.

example4:

```
>>> a,b = getDataPartners(var = 'U component of wind',Type = 'f',
          date = '2010-6-5',hour = 24,level = 'all',orginData = 1,
          datePriority = 'p', lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e. fcst 24 hour.its timeAxis date is '2010-6-6'. b is partnersData. i.e. anl. its anl date w.r.t orginData is '2010-6-5'. we can compare this eg4 with eg2. In this we passed datePriority as 'p'. So the passed date as set to the partnersData and orginData's date has shifted to the next day.

example5:

```
>>> a,b = getDataPartners(var = 'U component of wind',Type = 'a',
          date = ('2010-6-5','2010-6-6'),hour = 24,level = 'all',
    orginData = 1,datePriority = 'o', lat=(-90,90),lon=(0,359.5))
```

Note: Even though we passed 'a' Type, we must choose the hour option to select the fcst file, since we are passing the range of dates.

a is orginData. i.e. anl. its timeAxis size is 2. date are '2010-6-5' and '2010-6-6'.

b is partnersData. i.e. fcst 24 hour data. its timeAxis size is 2. date w.r.t orginData are '2010-6-4' and '2010-6-5'.

a's '2010-6-5' has partner is b's '2010-6-4'.i.e.orginData(anl) partners is partnersData's (fcst).

same concept for the remains day. a's '2010-6-6' has partner is b's '2010-6-5'.

example6:

```
>>> a,b = getDataPartners(var = 'U component of wind',Type = 'a',
           date = ('2010-6-5','2010-6-6'),hour = 24,level = 'all',
    orginData = 1,datePriority = 'p', lat=(-90,90),lon=(0,359.5))
```

Note: Even though we passed 'a' Type, we must choose the hour option to select the fcst file, since we are passing the range of dates.

a is orginData. i.e. anl. its timeAxis size is 2. date are '2010-6-6' and '2010-6-7'.

b is partnersData. i.e. fcst 24 hour data. its timeAxis size is 2. date w.r.t orginData are '2010-6-5' and '2010-6-6'.

a's '2010-6-6' has partner is b's '2010-6-5'.i.e.orginData(anl) partners is partnersData's (fcst).

same concept for the remains day. a's '2010-6-7' has partner is b's '2010-6-6'. we can compare this eg6 with eg5.In this we passed datePriority as 'p'. So the passed date as set to the partnersData and orginData's date has shifted towards the next days.

Written by: Arulalan.T

Date: 27.05.2011

`getMonthAvgData` (*var*, *Type*, *level*, *month*, *year*, *calendarName=None*, *hour=None*, ***latlonregion*)

`getMonthAvgData()`: It returns the average of the given month data (all days in the month) for the passed variable options

Condition : calendarName,level are optional. level takes default 'all' if not pass arg for it. hour is must when Type arg should be 'f' (fcst) to select xml object. Pass either (lat,lon) or region.

Inputs : Type - either 'a' or 'f' or 'o' var,level must be belongs to the data file month may be even in 3 char like 'apr' or 'April' or 'aPRiL' or like any month year must be passed as integer calendarName default None, it takes cdtime.DefaultCalendar key word arg lat,lon or region should be passed

Outputs : It should return the average of the whole month data for the given vars as MV2 variable

Usage :

example :

```
>>> getMonthAvgData(var = "Geopotential Height",Type = 'f',
                level = 'all', month = 'july',year=2010 , hour = 24,
                 region = AIR ) #lat=(-90,90),lon=(0,359.5)
```

returns dataAvg of one month data as single MV2 variable

Written by: Arulalan.T

Date: 29.04.2011

`getRainfallData` (*date=None*, *level='all'*, ***latlonregion*)

`getRainfallData()`: Extract the rainfall data from the xml file which has created by the cdscan command.

Inputs [var takes from the instance member of GribAccess class]

if we passed date,level then it should return data accordingly By default level takes 'all' levels. Pass either (lat,lon) or region keyword arg

Condition [we must set the member variable called rainfallXmlPath] and (rainfallXmlVar or rainfallModel), then only we can access this method. rainfallXmlVar is the obeservation rainfall variable name to access the data. OR rainfallModel is the model name, which has set in the global variable names settings. By Using it, this method should get the observation rainfall variable name.

Written by : Arulalan.T

Date : 29.05.2011

`getRainfallDataPartners`(*date*, *hour=None*, *level='all'*, *orginData=1*, *datePriority='o'*, ***latlonregion*)

`getRainfallDataPartners()`: It returns the rainfall data & its partners data(fcst is the partner of the observation. i.e. rainfall) as MV2 vars

Condition: startdate is must. enddata is optional one. If both startdate and enddate has passed means, it should return the rainfall data and partnersData within that range.

Inputs:

orginData - either 0 or 1. 0 means it shouldnot return the

orginData as single MV2 var. 1 means it should return both the orginData and its partnersData as two seperate MV2 vars.

datePriority - either 'o' or 'p'. 'o' means passed date is with respect to orginData. According to this orginData's date, it should return its partnersData. 'p' means passed date is with respect to partnersData.

According to this partnersData's date, it should return its orginData.

key word arg lat,lon or region should be passed By default hour is None and level is 'all'.

self.rainfallXmlPath is mandatory one when you choosed orginData is 1. you must set the rainfall xml path to the rainfallXmlPath.

self.rainfallModel is the model name, which has set in the global variables settings, to get the model fcst and its obeservation variable name to access the data. It is mandatory one.

date formate 1: date = (startdate,enddate) here startdate and enddate must be like cdtime.comptime formate.

date formate 2: date = (startdate)

date formate 3: date = 'startdate' or date = 'date'

eg for date input :

date = ('2010-5-1','2010-6-30') date = ('2010-5-30') date = '2010-5-30'

By default skipdays as 1 takes place. User cant override till now.

Outputs :

If user passed single date in the date argument, then it should return the data of that particular date (both orginData & partnersData) as MV2 variable.

If user passed start and enddate in the date argument, then it should return the data (both orginData & partnersData) for the range of dates as MV2 variable with time axis.

Usage :

Note: if 'r'(observation) file is orginData means 'f'(fcst) files are its partnersData.

example1:

```
>>> a,b = getRainfallDataPartners(date = '2010-6-5',hour = None,
                 level = 'all',orginData = 1,datePriority = 'o',
                                  lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e. rainfall observation. its timeAxis date is ‘2010-6-5’.

b is partnersData. i.e. fcst. its 24 hour fcst date w.r.t orginData is ‘2010-6-4’. 48 hour is ‘2010-6-3’.

Depends upon the availability of date of fcst files,it should return the data. In NCMRWF2010 model, it should return maximum of 7 days fcst.

If we will specify any hour in the same eg, that should return only that hour fcst file data instead of returning all the available fcst hours data.

example2:

```
>>> a,b = getRainfallDataPartners(date = '2010-6-5',hour = 24,
                level = 'all',orginData = 1,datePriority = 'o',
                                    lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e. rainfall observation. its timeAxis date is ‘2010-6-5’. b is partnersData. i.e. fcst 24 hour. its fcst date w.r.t orginData is ‘2010-6-6’.

example3:

```
>>> b = getRainfallDataPartners(date = '2010-6-5',hour = 24,
                level = 'all',orginData = 0,datePriority = 'o',
                                    lat=(-90,90),lon=(0,359.5))
```

b is partnersData. i.e. fcst. its fcst date w.r.t orginData is ‘2010-6-6’. No orginData. Because we passed orginData as 0.

example4:

```
>>> a,b = getRainfallDataPartnerss(date = '2010-6-5',hour = 24,
                level = 'all',orginData = 1,datePriority = 'p',
                                    lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e. rainfall observation. its timeAxis date is ‘2010-6-6’.

b is partnersData. i.e. fcst 24 hour. its fcst date w.r.t orginData is ‘2010-6-5’. we can compare this eg4 with eg2.

In this we passed datePriority as ‘p’. So the passed date as set to the partnersData and orginData’s date has shifted to the next day.

example5:

```
>>> a,b = getRainfallDataPartners(date = ('2010-6-5','2010-6-6'),
                          hour = 24,level = 'all',orginData = 1,
                datePriority = 'o', lat=(-90,90),lon=(0,359.5))
```

Note: We must choose the hour option to select the fcst file, since we are passing the range of dates.

a is orginData.i.e.rainfall observation.its timeAxis size is 2. date are ‘2010-6-5’ and ‘2010-6-6’.

b is partnersData.i.e.fcst 24 hour data.its timeAxis size is 2. date w.r.t orginData are ‘2010-6-4’ and ‘2010-6-5’.

a's '2010-6-5' has partner is b's '2010-6-4'. i.e. orginData(rainfall observation) partners is partnersData(fcst)

same concept for the remains day. a's '2010-6-6' has partner is b's '2010-6-5'.

example6:

```
>>> a,b = getRainfallDataPartners(date = ('2010-6-5','2010-6-6'),
                              hour = 24,level = 'all',orginData = 1,
                      datePriority = 'p', lat=(-90,90),lon=(0,359.5))
```

a is orginData. i.e. rainfall observation. its timeAxis size is 2. date are '2010-6-6' and '2010-6-7'.

b is partnersData. i.e. fcst 24 hour data. its timeAxis size is 2. date w.r.t orginData are '2010-6-5' and '2010-6-6'.

a's '2010-6-6' has partner is b's '2010-6-5'. i.e. orginData(rainfall observation) partners is partnersData(fcst)

same concept for the remains day. a's '2010-6-7' has partner is b's '2010-6-6'. we can compare this eg6 with eg5. In this we passed datePriority as 'p'. So the passed date as set to the partnersData and orginData's date has shifted towards the next days.

Written by: Arulalan.T

Date: 29.05.2011

getXmlPath(*Type*, *hour=None*)

`getXmlPath()`: To get the xml's absolute path.

Inputs [Type is either 'a' or 'o' or 'f' or 'r'] hour is mandatory when you pass 'f' or 'r'. 'a' - analysis, 'f' - forecast, 'o' - observation, 'r' - reference.

Written by : Arulalan.T

Date : 21.08.2011

listvariable(*Type*, *hour=None*)

:func:'listvariable': By passing Type and/or hour args to this method, it will return the listvariable method of the appropriate xml file.

Returns the listvariable of cdms2 open object method result

2.2 Time Axis Utils

This timeutils module helps us to generate our own time axis, correct existing time axis bounds and generate bounds.

Here we used inbuilt methods of cdtime and cdutil module of uv-cdat.

2.2.1 timeutils

2.3 Plot Utils

The plot module has the properties to plot the vcs vector with some default template look out.

User can control the reference point, scale of arrow marks of the vector plot.

2.3.1 plot

class `plot.`**`reference_std_dev`**

This class implements the type: standard deviation of a reference variable. It is just a float value with one method that will be used in the computation of a test variable RMS.

`compute_RMS_function`(*s*, *R*)

Compute and return the centered-pattern RMS of a test variable from its standard-deviation and correlation.

Input: self: reference-variable standard deviation s: test variable standard deviation R: test-variable correlation with reference variable

`plot.`**`vectorPlot`**(*u*, *v*, *name*, *path=None*, *reference=20.0*, *scale=1*, *interval=1*, *svg=1*, *png=0*, *latlabel='lat5'*, *lonlabel='lon5'*, *style='portrait'*)

`vectorPlot()`: Plotting the vector with some default preferences.

Input [u - u variable] v - v variable name - name to plot on the top of the vcs path - path to save as the image file. reference - vector reference. Default it takes 20.0 (i.e 2 degree) scale - scaling of the arrow mark in vector plot interval - slicing the data to reduce the density (noise)

in the vector plot, with respect to the interval. Default it takes 1. (i.e. doesnot affect the u & v)

svg - to save image as svg png - to save image as png

Condition [u must be 'u variable' and v must be 'v variable'.] name must pass to set the name on the vector vcs path is not passed means, it takes current workig directory reference must be float. interval not be 0.

Usage : using this function, user can plot the vector.

user can control the reference point of the vector, and scale length of the arrow marks in plot.

Also can control the u and v data shape by interval.

filename should be generated from the 'name' passed by the user, just replacing the space into underscore '_'.

if svg and png passed 1, the image will be saved with these extensions in the filename.

Written By : Arulalan.T

Date : 26.07.2011

2.4 More

More utilities will be added and optimized in near future.

CHAPTER

THREE

DOCUMENTATION OF DIAGNOSIS SOURCE CODE

The diagnosis package contains the following modules.

- Daily Progress
- Monthly Progress
- Seasonly Progress
- Statistical Scores
- More

3.1 Daily Progress

The daily progress of diagnosis are listed below.

3.1.1 Anomaly

Have to update the code. Will do it soon.

3.2 Monthly Progress

The monthly progress of diagnosis are listed below.

- Month Mean
- Month Anomaly
- Month Fcst Sys Error

These monthly progress will be automated.

3.2.1 Month Mean

The word *Mean* means average of the data. The average will be taken over the month time axis is called month mean.

The below script *compute_month_mean.py* should explain more how we are implementing monthly mean and generating the nc files.

`compute_month_mean.`**`genMonthMeanDirs`**(*modelname*, *modelpath*, *modelhour*)

genMonthMeanDirs() [It should generate the directory structure] whenever it needs. It reads the timeAxis information of the model data xml file(which is updating it by cdscan), and once the full months is completed, then it should check either that month directory is empty or not.

case 1: If that directory is empty means, it should call the function called *genMonthMeanFiles*, to calculate the mean analysis and anomaly for that month and should store the processed files in side that directory.

case 2: If that directory is non empty means, **have to update***

Inputs [modelname is the model data name, which will become part of the] directory structure. modelpath is the absolute path of data where the model xml files are located. climatolgyyear is the year of climatolgy data. climregridpath is the absolute path of the climatolgy regridded path w.r.t to this model data resolution (both horizontal and vertical) climpfilename is the climatolgy Partial File Name to combine the this passed name with (at the end) of the climatolgy var name to open the climatolgy files.

Outputs [It should create the directory structure in the processfilesPath] and create the processed nc files.

Written By : Arulalan.T

Date : 01.12.2011

`compute_month_mean.`**`genMonthMeanFiles`**(*meanMonthPath*, *monthdate*, *year*, *typehour*, ***model*)

genMonthMeanFiles() [It should calculate monthly mean analysis &] monthly mean forecast hours value for the month (of year). Finally stores it as nc files in corresponding directory path which are passed in this function args.

Inputs [meanMonthPath is the absolute path where the processed month mean] analysis & fcst hour nc files are going to store. monthdate (which contains monthname, startdate & enddate) and year are the inputs to extract the monthly data. typehour is tuple which has the type key character and fcst hour to create sub directories inside mean directory.

KWargs: modelName, modelXmlPath, modelXmlObj

modelName is the model data name which will become part of the process nc files name. modelPath is the absolute path of data where the model xml files are located. modelXmlObj is an instance of the GribXmlAccess class instance. If we are passing modelXmlObj means, it will be optimized one when we calls this same function for same model for different months.

We can pass either modelXmlPath or modelXmlObj KWarg is enough.

Outputs [It should create monthly mean analysis and monthly mean forecast] hours for all the available variables in the vars.txt file & store it as nc file formate in the proper directories structure (modelname, process name, year, month and then [Analysis or hours] hierarchy).

Written By : Arulalan.T

Date : 08.09.2011 Updated : 06.12.2011

3.2.2 Month Anomaly

Hello This is math test. Remove me ! $a^2 + b^2 a\_b = c^2$.

$$(a+b)^2 = a^2 + 2ab + b^2$$
$$(a-b)^2 = a^2 - 2aba\_b + b^2$$

Anomaly means the difference between the model analysis and climatology.

Monthly Anomaly : Take the difference between the model analysis data of the particular month and the climatology data of the corresponding month.

Anomaly = Analysis - Climatology

The below script *compute_month_anomaly.py* should explain more how we are implementing monthly anomaly and generating the nc files.

`compute_month_anomaly.`**`genMonthAnomalyDirs`**(*modelname*, *modelpath*, *climregridpath*, *climpfilename*, *climatologyyear*)

`genMonthAnomalyDirs()` [It should generate the directory structure] whenever it needs. It reads the timeAxis information of the model data xml file(which is updating it by cdscan), and once the full months is completed, then it should check either that month directory is empty or not.

case 1: If that directory is empty means, it should call the function called *genMonthAnomalyFiles*, to calculate the mean analysis and anomaly for that month and should store the processed files in side that directory.

case 2: If that directory is non empty means, **have to update***

Inputs [modelname is the model data name, which will become part of the] directory structure. modelpath is the absolute path of data where the model xml files are located. climregridpath is the absolute path of the climatolgy regridded path w.r.t to this model data resolution (both horizontal and vertical) climpfilename is the climatolgy Partial File Name to combine the this passed name with (at the end) of the climatolgy var name to open the climatolgy files. climatolgyyear is the year of climatolgy data.

Outputs [It should create the directory structure in the processfilesPath] and create the processed nc files.

Written By : Arulalan.T

Date : 01.12.2011

`compute_month_anomaly.`**`genMonthAnomalyFiles`**(*meanAnomalyPath*, *meanAnalysisPath*, *climRegridPath*, *climPFileName*, *climatologyYear*, *monthdate*, *year*, ***model*)

`genMonthAnomalyFiles()` [It should calculate monthly mean anomaly] from the monthly mean analysis and monthly mean climatolgy, for the month (of year) and process it. Finally stores it as nc files in corresponding directory path which are passed in this function args.

Inputs [meanAnomalyPath is the absolute path where the processed mean] anomaly nc files are going to store. meanAnalysisPath is the absolute path where the processed mean analysis nc files were already stored. climRegridPath is the absolute path where the regridded monthly mean climatologies (w.r.t the model vertical resolution) nc files were already stored. climPFileName is the partial nc filename of the climatolgy. climatologyYear is the year of the climatolgy to access it. monthdate (which contains monthname, startdate & enddate) and year are the inputs to extract the monthly data.

KWargs: modelName, modelXmlPath, modelXmlObj

modelName is the model data name which will become part of the process nc files name. modelPath is the absolute path of data where the model xml files are located. modelXmlObj is an instance of the GribXmlAccess class instance. If we are passing modelXmlObj means, it will be optimized one when we calls this same function for same model for different months.

We can pass either modelXmlPath or modelXmlObj KWarg is enough.

Outputs [It should create mean anomaly for the particular variables which] are all set the clim_var option in the vars.txt file. Finally store it as nc file formate in the proper directories structure (modelname, process name, year and then month hierarchy).

Written By : Arulalan.T

Date : 08.09.2011 Updated : 07.12.2011

3.2.3 Month Fcst Sys Error

Forecast Systematic Error means the difference between the model forecast hour data and model analysis.

This also called as *Fcst Sys Err*.

Month Fcst Sys Error : Take the difference between the model forecast hour data of the particular month and the model analysis data of the same month.

Fcst Sys Err = Model Fcst Hour Data - Model Analysis

The below script *compute_month_fcst_sys_error.py* should explain more how we are implementing monthly fcst sys err and generating the nc files.

`compute_month_fcst_sys_error.`**`genMonthFcstSysErrDirs`**(*modelname*, *modelpath*, *modelhour*)

`genMonthFcstSysErrDirs()` [It should generate the directory structure] whenever it needs. It reads the timeAxis information of the model data xml file(which is updating it by cdscan), and once the full months is completed, then it should check either that month directory is empty or not.

case 1: If that directory is empty means, it should call the function called *genMonthFcstSysErrFiles*, to calculate the mean analysis and fcstsyserr for that month and should store the processed files in side that directory.

case 2: If that directory is non empty means, **have to update***

Inputs [modelname is the model data name, which will become part of the] directory structure. modelpath is the absolute path of data where the model xml files are located. modelhour is the list of model data hours, which will become part of the directory structure.

Outputs [It should create the directory structure in the processfilesPath] and create the processed nc files.

Written By : Arulalan.T

Date : 08.12.2011

`compute_month_fcst_sys_error.`**`genMonthFcstSysErrFiles`**(*meanFcstSysErrPath*, *meanPath*, *monthdate*, *year*, *modelhour*, ***model*)

`genMonthFcstSysErrFiles()` [It should calculate mean analysis &] fcstsyserr for the passed month (of year) and process it. Finally stores it as nc files in corresponding directory path which are passed in this function args.

Inputs [meanFcstSysErrPath is the absolute path where the processed mean] fcstsyserr nc files are going to store. meanPath is the absolute path (partial path) where the processed monthly mean analysis and fcst hour nc files were stored already. monthdate (which contains monthname, startdate & enddate) and year are the inputs to extract the monthly data. modelhour is the list of model data hours, which will become part of the directory structure.

KWargs: modelName, modelXmlPath, modelXmlObj

modelName is the model data name which will become part of the process nc files name. modelPath is the absolute path of data where the model xml files are located. modelXmlObj is an instance of the GribXmlAccess class instance. If we are passing modelXmlObj means, it will be optimized one when we calls this same function for same model for different months.

We can pass either modelXmlPath or modelXmlObj KWarg is enough.

Outputs [It should create mean forecast systematic error for all the] available variables in the vars.txt file & store it as nc file formate in the proper directories structure (modelname, process name, year, month and then hours hierarchy).

Written By : Arulalan.T

Date : 08.09.2011 Updated : 08.12.2011

3.3 Seasonly Progress

The seasonly progress of diagnosis are listed below.

- Season Mean
- Collect Season Fcst Rainfall
- Region Statistical Score
- Season Statistical Score Spatial Distribution

These season progress will be automated with respect to the given season as input in the configure.txt.

The word *season* means the consecutive months.

For eg: JJAS contains June, July, August, September.

3.3.1 Season Mean

The word *Mean* means average of the data. The average will be taken over the season time axis is called season mean.

The below script *compute_season_mean.py* should explain more how we are implementing seasonly mean and generating the nc files.

`compute_season_mean.`**`genMeanAnlFcstErrDirs`**(*modelname*, *modelpath*, *modelhour*)

`genMeanAnlFcstErrDirs()` [It should create the directory structure] whenever it needs. It reads the timeAxis information of the model data xml file(which is updating it by cdscan), and once the full seasonal months are completed, then it should check either that season directory is empty or not.

case 1: If that directory is empty means, it should call the function called *genSeasonMeanFiles*, to calculate the mean analysis and fcstsyserr for that season and should store the processed files in side that directory.

case 2: If that directory is non empty means, **have to update***

Inputs [modelname is the model data name, which will become part of the] directory structure. modelpath is the absolute path of data where the model xml files are located. modelhour is the list of model data hours, which will become part of the directory structure.

Outputs [It should create the directory structure in the processfilesPath] and create the processed nc files.

Written By : Arulalan.T

Date : 08.12.2011

`compute_season_mean.`**`genSeasonMeanFiles`**(*meanSeasonPath*, *meanMonthPath*, *seasonName*, *seasonMonthDate*, *year*, *Type*, ***model*)

`genSeasonMeanFiles()` [It should calculate the seasonly mean for] either analysis or forecast systematic error. It can be choosed by the Type argment. Finally stores it as nc files in corresponding directory path which are passed in this function args.

Inputs [meanSeasonPath is the absolute path where the processed season] mean analysis or forecast systematic error nc files are going to store. Inside the fcst hours directories will be created in this path, if needed.

meanMonthPath is the absolute path where the processed monthly mean analysis or monthly mean fcstsyserr nc files were stored, already. seasonName is the name of the season. seasonMonthDate(list of months which contains monthname, startdate & enddate) for the season. year is the part of the directory structure. Type is either 'a' for analysis or 'f' for fcstsyserr.

KWargs: modelName, modelXmlPath, modelXmlObj

modelName is the model data name which will become part of the process nc files name. modelHour is the model hours as list, which will become part of the directory structure. modelPath is the absolute path of data where the model xml files are located. modelXmlObj is an instance of the GribXmlAccess class instance. If we are passing modelXmlObj means, it will be optimized one when we calls this same function for same model for different months.

We can pass either modelXmlPath or modelXmlObj KWarg is enough.

Process [This function should compute the seasonly mean for analysis and] fcstsyserr by just opening the monthly mena analysis/fcstsyserr nc files (according to the season's months) and multiply the monthly mean into its weights value. So that monthly mean data should become monthly full data (not mean). Then add it together for the season of months. Finally takes the average by just divide the whole season data by sum of monthly mean weights.

So it should simplify our life, just extracting data which

timeAxis length is 4, for 4 months in season. (eg JJAS).

Outputs [It should create seasonly mean analysis and forecast systematic] error for all the available variables in the vars.txt file, and store it as nc file in the proper directory structure (modelname, process name, year, season, and/or hours hierarchy).

Written By : Arulalan.T

Date : 08.12.2011

3.3.2 Collect Season Fcst Rainfall

This script *collect_season_fcst_rainfall.py* should collect the whole season forecast hourly rainfall with respect to the hours of season. Finally it should create the nc files for every fcst hours that contains the fcst rainfall with needed time axis to compute the further process.

Written by: Dileepkumar R JRF- IIT DELHI

Date: 23.06.2011

Updated By : Arulalan.T Date: 14.09.2011 Date: 19.10.2011

`collect_season_fcst_rainfall.`**`createSeaonFcstRainfallData`**(*modelname*, *modelpath*, *modelhour*, *rainfallPath*, *rainfallXmlName=None*)

`createSeaonFcstRainfallData()`: It should create model hours forecast rainfall data nc files, in side the 'StatiScore' directory of processfilesPath in hierarchy structure. The fcst rainfall timeAxis are in partners timeAxis w.r.t observation rainfall and fcst hours.

3.3.3 Region Statistical Score

The below script *compute_region_statistical_score.py* should compute the various statistical scores regional wise and make the nc files.

Here regions are 'Central India', 'Peninsular India', etc. That is region name/variable defines the latitude and longitude range.

Example: Let 'ts'(Threat Score) is a statistical score, we are calculate this for different regions.

Written by: Dileepkumar R JRF- IIT DELHI

Date: 02.08.2011;

Updated By : Arulalan.T Date : 16.09.2011

`compute_region_statistical_score.`**`genStatisticalScore`**(*modelname*, *modelhour*, *seasonName*, *year*, *procStatiSeason*, *procRegion*, *plotCSV*, *rainfallPath*, *rainfallXmlName=None*)

`genStatisticalScore()` [It should compute the statistical scores] like " Threat Score, Equitable Threat Score, Accuracy(Hit Rate), Bias Score, Probability Of Detection, False Alarm Rate, Odds Ratio, Probability Of False Detection, Kuipers Skill Score, Log Odd Ratio, Heidke Skill Score, Odd Ratio Skill Score, & Extreame Dependency Score" by accessing the observation and forecast data.

It should compute the statistical scores for different regions.

Finally it should store the scores variable in both csv and nc files in appropriate directory hierarchy structure.

..note:: We are replacing the -ve values with zeros of both the observation and fcst data to make correct statistical scores.

Inputs [modelname, modelhour, seasonName, year are helps to generate the] path. procStatiSeason is the partial path of process statistical score season path. procRegion is an absolute path to store the nc files plotCSV is an absolute path to store the csv files. rainfallPath is the path of the observation rainfall. rainfallXmlName is the name of the xml file name, it is an optional one. By default it takes 'rainfall_regrided.xml'

Outputs [It should store the computed statistical scores for all the] regions and store it as both ncfile and csv files in the appropriate directory hierarchy structure.

`compute_region_statistical_score.`**`genStatisticalScoreDirs`**(*modelname*, *modelhour*, *rainfallPath*, *rainfallXmlName=None*)

Func *genStatisticalScoreDirs* : It should generate the appropriate directory hierarchy structure for 'StatiScore' in both the processfiles path and plotgraph path. In plotgraph path, it should create 'CSV' directory to store the 'statistical scores' in csv file formate.

This function should call the *genStatisticalScore* function to compute and statistical score.

Inputs [modelname and modelhour are the part of the directory hierarchy] structure. rainfallPath is the path of the observation rainfall. rainfallXmlName is the name of the xml file name.

3.3.4 Season Statistical Score Spatial Distribution

The below script *compute_season_stati_score_spatial_distribution.py* should compute the various statistical scores w.r.t each & every lat, lon location of particular region.

Example: Let 'TS'(Threat Score) is a statistical score, we are calculate this spatially.

Written by: Dileepkumar R JRF- IIT DELHI

Date: 02.09.2011;

Updated By : Arulalan.T Date : 17.09.2011 Date : 06.10.2011

`compute_season_stati_score_spatial_distribution.`**`genStatisticalScorePath`**(*modelname*, *modelhour*, *rainfallPath*, *rainfallXmlName=None*)

`genStatisticalScorePath()`: It should make the existing path of process files statistical score. Also if that is correct path means, it should calls the function *genStatisticalScoreSpatialDistribution* to compute the statistical score in spatially distributed way.

Inputs [modelname and modelhour are the part of the directory hierarchy] structure.

`compute_season_stati_score_spatial_distribution.`**`genStatisticalScoreSpatialDistribution`**(*mode*
modelhour
season,
year,
statiSea
sonPath
rainfallPath
rainfallXml
Nam
lat=
lon=

Func *genStatisticalScoreSpatialDistribution* : It should compute the statistical scores like " Threat Score, Equitable Threat Score, Accuracy(Hit Rate), Bias Score, Probability Of Detection, Odds Ratio, False Alarm Rate, Probability Of False Detection, Kuipers Skill Score, Log Odd Ratio, Heidke Skill Score, Odd Ratio Skill Score, & Extreame Dependency Score" in spatially distributed way (i.e compute scores in each and every lat & lon points) by accessing the observation and forecast data.

Inputs [modelname, modelhour, season, year are helps to generate the] path. statiSeasonPath is the partial path of process statistical score season path. rainfallPath is the path of the observation rainfall. rainfallXmlName is the name of the xml file name, it is an optional one. By default it takes 'rainfall_regrided.xml'. lat, lon takes tuple args. If we passed it, then the model lat, lon should be shrinked according to the passed lat,lon. Some times it may helpful to do statistical score in spatially distributed in particular region among the global lat,lon.

Outputs [It should store the computed statistical scores in spatially] distributed way for all the modelhour(s) as nc files in the appropriate directory hierarchy structure.

3.4 Diagnosis Plots

The diagnosis plots of diagnosis are listed below.

- Winds Plots
- Iso Plots
- Statistical Score Bar Plots
- Statistical Score Spatial Distribution Plots

3.4.1 Winds Plots

Generate the vector plots using U and V component of the wind data. The below script *generate_winds_plots.py* should generates the wind plots and save it as either png or jpg or svg in the suitable directory for month and season wise.

`generate_winds_plots.`**`editVectorPlot`**(*modelname*, *processtype*, *year*, *monthseason*, *hour=None*, *level=None*, *region=None*, *reference=20*, *scale=1*, *interval=4*, *svg=0*, *png=1*, *latlabel='lat5'*, *lonlabel='lon5'*, *outpath=None*)

`editVectorPlot()` [To edit/reproduce any particular vector plots by] passing modelname, processtype, year, month/season, hour, level(s), region, reference point of vector, scale of vector arrow markers, interval of the U & V datasets, svg & png options.

Inputs [modelname is the part of the directory structure.] processtype is any one of the processes.for eg : 'Mean Analysis', 'Mean Fcst' or 'Anomaly' or 'FcstSysErr', etc., year is year in string type. monthseason is either month name or season name. It should find out either it is month or season and make the correct path. hour is the hour string which is the part of the directory structure only for 'FcstSysErr' and 'Mean Fcst' process type.

level is either single level, or list of levels or 'all'. 'all' means, it takes all the availableLevels from the variables. level value must be int, float only. Not be string, other than 'all' keyword. region is the region variable which should cut particular region shape from the global data. By default it is None, i.e. takes global data region itself.

reference is the reference points to be plotted in the vcs vector plot. It must be float only. scale is the length of the arrow markers in the vcs vector plot. interval is the integer value, to split the U and V datasets, to make clear view of the vector plot. By default it takes 4.

svg is the flag. If flag is set, then the vector plot should be saved as svg formate. By default it is 0.

png is the flag. If flag is set, then the vector plot should be saved as png formate. By default it is 1.

outpath is the absolute path, where the generated plots should be stored. By default, it is None, that is it should save in the appropirate plotsgraphs directory, which is generated by this function.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

`generate_winds_plots.`**`genMonthAnomalyDirs`**(*modelName*, *availableMonths*)

`genMonthAnomalyDirs()`: It should generate the directory hierarichy structure of month anomaly in the plotsgraphspath. And calls the function genVectorPlots to make vector plots and save it inside the appropirate directory, by reading the u, v nc files of the appropirate process month anomaly files path.

Inputs [modelName is the one of the directories name.] availableMonths is the dictionary which is generated by fully available months from the timeAxis.

..note:: It should takes the levels which is set in the global config file, and generate the vector plots to those levels only.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

`generate_winds_plots.`**`genMonthMeanDirs`**(*modelName*, *availableMonths*)

`genMonthMeanDirs()`: It should generate the directory hierarichy structure of month mean in the plotsgraphspath. And calls the function genVectorPlots to make vector plots and save it inside the appropirate directory, by reading the u, v nc files of the appropirate process month mean files path.

Inputs [modelName is the one of the directories name.] availableMonths is the dictionary which is generated by fully available months from the timeAxis.

..note:: It should takes the levels which is set in the global config file, and generate the vector plots to those levels only.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

`generate_winds_plots.`**`genSeasonFcstSysErrDirs`**(*modelName*, *modelHour*, *availableMonths*)

`genSeasonFcstSysErrDirs()`: It should generate the directory hierarichy structure of season fcstsyserr in the plotsgraphspath. And calls the function genVectorPlots to make vector plots and save it inside the appropirate directory, by reading the xml file of the appropirate process season fcstsyserr files path.

Inputs [modelName is the one of the directories name.] modelHour is the one of the directories name. availableMonths is the dictionary which is generated by fully available months from the timeAxis.

..note:: It should takes the levels which is set in the global config file, and generate the vector plots to those levels only.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

`generate_winds_plots.`**`genSeasonMeanDirs`**(*modelName*, *availableMonths*)

`genSeasonMeanDirs()`: It should generate the directory hierarichy structure of season mean in the plotsgraphspath. And calls the function genVectorPlots to make vector plots and save it inside the appropirate directory, by reading the xml file of the appropirate process season mean files path.

Inputs [modelName is the one of the directories name.] availableMonths is the dictionary which is generated by fully available months from the timeAxis.

..note:: It should takes the levels which is set in the global config file, and generate the vector plots to those levels only.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

`generate_winds_plots.`**`genVectorPlots`**(*uvar*, *vvar*, *upath=None*, *vpath=None*, *xmlpath=None*, *outpath=None*, *month=None*, *date=None*, *level=None*, *region=None*, *reference=20.0*, *scale=1*, *interval=4*, *svg=0*, *png=1*, *latlabel='lat5'*, *lonlabel='lon5'*, *style='portrait'*)

`genVectorPlots()`: It should generate the vector plots in vcs background and save it as png(by default) inside the outpath, with some default vector properties like reference, scale and interval.

Inputs [uvar is the 'u' variable name] vvar is the 'v' variable name upath is the 'u' nc file absolute path. vpath is the 'v' nc file absolute path. xmlpath is the xml file absolute path which must contains the u and v vars. outpath is the absolute path, where the generated plots should be stored. By default, it is None. It means, it should save in the current working directory itself. level is either single level, or list of levels or 'all'. 'all' means, it takes all the availableLevels from the variables. level value must be int, float only. Not be string, other than 'all' keyword. region is the region variable which should cut particular region shape from the global data. By default it is None, i.e. takes global data region itself.

reference is the reference points to be plotted in the vcs vector plot. It must be float only. scale is the length of the arrow markers in the vcs vector plot. interval is the integer value, to split the U and V datasets, to make clear view of the vector plot. By default it takes 4.

svg is the flag. If flag is set, then the vector plot should be saved as svg formate. By default it is 0.

png is the flag. If flag is set, then the vector plot should be saved as png formate. By default it is 1.

Condition [If we passed xmlpath, then we no need to pass upath and vpath] args. uvar and vvar must be available in the passed filepath.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

`generate_winds_plots.`**`getProcessPath`**(*modelname*, *processtype*, *year*, *monthseason*, *hour=None*)

`getProcessPath()`: By passing fewer args and get the correct and absolute path of the process files, which generated by automated or manual for the purpose of this diagnosis.

Inputs [modelname is the part of the directory structure.] processtype is any one of the processes.for eg : 'Mean Analysis', or 'Mean Fcst' or 'Anomaly' or 'FcstSysErr', etc., year is year in string type. monthseason is either month name or season name. It should find out either it is month or season and make the correct path. hour is the hour string which is the part of the directory structure only for 'FcstSysErr' and 'Mean Fcst' process type.

Outputs [Return the absolute path of the process files, only if that] directory is exists. Other wise it raise error.

To Do [Need to decide about, either it should raise error, or it should] return None, if wrong args passed or process directory doesnot exists.

Written By : Arulalan.T

Date : 11.09.2011 Updated: 11.12.2011

3.4.2 Iso Plots

Generate the iso plots for the iso variables which are all set in the 'vars.txt' file.

The below script *generate_iso_plots* should generates the following kind of plots.

- iso line
- iso fill

- iso fill line

Finally save the generated iso plots as either png or jpg or svg in the suitable directory for month and season wise.

`generate_iso_plots.`**`genIsoFillLinePlots`**(*var*, *key*, *isoLevels*, *xmlpath=None*, *path=None*, *outpath=None*, *month=None*, *date=None*, *level='all'*, *region=None*, *svg=0*, *png=1*)

`genIsoFillLinePlots()`: It should generate the isoFillLine plots in vcs background and save it as png(by default) inside the outpath, with isoLevels (passed by user) and isoColors (default).

Inputs [var is the variable name.] key to identify, it is which variable to make plot name. isoLevels is the levels to plot isoFillLine and set the legend levels in vcs. xmlpath is the xml file absolute path. path is the nc file absolute path. pass any one (path or xmlpath)

outpath is the absolute path, where the generated plots should be stored. By default, it is None. It means, it should save in the current working directory itself. level is either single level, or list of levels or 'all'. 'all' means, it takes all the availableLevels from the variables. level value must be int, float only. Not be string, other than 'all' keyword. region is the region variable which should cut particular region shape from the global data. By default it is None, i.e. takes global data region itself.

svg is the flag. If flag is set, then the vector plot should be saved as svg formate. By default it is 0.

png is the flag. If flag is set, then the vector plot should be saved as png formate. By default it is 1.

Written By : Arulalan.T

Date : 21.09.2011 Updated: 12.12.2011

`generate_iso_plots.`**`genIsoLinePlots`**(*var*, *key*, *xmlpath=None*, *path=None*, *outpath=None*, *month=None*, *date=None*, *level='all'*, *region=None*, *svg=0*, *png=1*)

`genIsoLinePlots()`: It should generate the isoLine plots in vcs background and save it as png(by default) inside the outpath, with isoLevels (find out by this method) and isoColors (default).

Inputs [var is the variable name.]

key to identify, it is which variable to make plot name. xmlpath is the xml file absolute path. path is the nc file absolute path. pass any one (path or xmlpath)

outpath is the absolute path, where the generated plots should be stored. By default, it is None. It means, it should save in the current working directory itself. level is either single level, or list of levels or 'all'. 'all' means, it takes all the availableLevels from the variables. level value must be int, float only. Not be string, other than 'all' keyword. region is the region variable which should cut particular region shape from the global data. By default it is None, i.e. takes global data region itself.

svg is the flag. If flag is set, then the vector plot should be saved as svg formate. By default it is 0.

png is the flag. If flag is set, then the vector plot should be saved as png formate. By default it is 1.

..note:: This function should find out the isoLevels to set in the isoline plot and its legend in vcs. IsoLevels is the range of min and max of all the levels data min and max.

Written By : Arulalan.T

Date : 21.09.2011 Updated: 12.12.2011

`generate_iso_plots.`**`genSeasonFcstSysErrDirs`**(*modelName*, *modelHour*, *availableMonths*, *plotLevel*)

`genSeasonFcstSysErrDirs()`: It should generate the directory hierarichy structure of season fcst-syserr in the plotsgraphspath. And calls the function genIsoFillLinePlots to make 'isofillline' plots and save it inside the appropirate directory, by reading the xml file of the appropirate process season fcstsyserr files path.

It should plot for all the vars in the 'isovars' which has set in the global 'vars.txt' file.

To plot isoFillLinePlot, this function should find out the isoLevels for all the variables of all the levels and all the hours.

isoLevels [It is a range of levels which is from the min (of data of] all the hours and levels), to the max (of data of all the hours and levels) to set the levels in the vcs plot and legend.

Inputs [modelName is the one of the directories name.] modelHour is the one of the directories name. availableMonths is the dictionary which is generated by fully available months from the timeAxis.

..note:: It should takes the levels which is set in the global config file, and generate the 'IsoFillLine' plots to those levels only.

Written By : Arulalan.T

Date : 20.09.2011 Updated: 12.12.2011

`generate_iso_plots.`**`genSeasonMeanDirs`**(*modelName*, *availableMonths*, *plotLevel*)

`genSeasonMeanDirs()`: It should generate the directory hierarichy structure of season mean in the plotsgraphspath. And calls the function genIsoLinePlots to make 'isoline' plots and save it inside the appropirate directory, by reading the xml file of the appropirate process season mean files path.

Inputs [modelName is the one of the directories name.] availableMonths is the dictionary which is generated by fully available months from the timeAxis.

..note:: It should takes the levels which is set in the global config file and generate the 'isoline' plots to those levels only.

Written By : Arulalan.T

Date : 20.09.2011 Updated: 12.12.2011

3.4.3 Statistical Score Bar Plots

The script *generate_statistical_score_bars.py* should generate the bar plots w.r.t the statistical scores for different region & fcst hours. Finally save the generated bar plots as either png or jpg or svg in the suitable directory for season wise. Date : 04.08.2011

Updated on : 28.09.2011

`generate_statistical_score_bars.`**`genBarDiagrams`**(*var*, *path*, *hours*, *outpath=None*, *bargap=0.28*, *barwidth=0.8*, *yticdiff=0.25*)

`genBarDiagrams()`: It should generate the least directory hierarichy structure of season statiscore in the plotsgraphspath by score name. It will plots score values in xmgrace as bar diagram and save it inside the appropirate directory, by reading the nc file of the appropirate process season Region statiscore files path.

It should plot for all the vars of that statiscore nc files.

Inputs [var is the variable name. If var is 'all' means, then it should] plot the bar diagram for all the available variables in the passed path nc or xml file.

path is an absolute nc or xml file path.

outpath is the path to store the images. If it is None means, it should create the least (plotname)directory in the current directory path itself and save it.

bargap is the value of the gap ratio in between each bars of each threshold in xaxis of score bar diagram.

barwidth is the width of the each bar in xaxis of the score bar diagram.

yticdiff is the difference of the tic levels in y axis of the bar diagram.

Written By : Dileep Kumar.R, Arulalan.T

Updated on : 28.09.2011

`generate_statistical_score_bars.`**`genSeasonStatiScoreDirs`**(*modelName*, *modelHour*, *availableMonths*)

`genSeasonStatiScoreDirs()`: It should generate the directory hierarichy structure of season statiscore in the plotsgraphspath. And calls the function genIsoFillPlots to make 'isofill' plots and save it inside the appropirate directory, by reading the nc file of the appropirate process season hour statiscore files path.

It should plot for all the vars of that statiscore spatial distributed nc files.

Inputs [modelName is the one of the directories name.] modelHour is the one of the directories name. availableMonths is the dictionary which is generated by fully available months from the timeAxis.

Written By : Arulalan.T

Date : 27.09.2011

3.4.4 Statistical Score Spatial Distribution Plots

The script *generate_stati_score_spatial_distribution_plots.py* should generate the iso fill plots w.r.t the statistical scores for different fcst hours. Finally save the generated iso fill plots as either png or jpg or svg in the suitable directory for season wise.

`generate_stati_score_spatial_distribution_plots.`**`genIsoFillPlots`**(*var*, *path*, *outpath=None*, *region=None*, *svg=0*, *png=1*)

`genIsoFillPlots()`: It should generate the directory least hierarichy structure of season statiscore in the plotsgraphspath,by the plotname.

It should plot for all the vars of that statiscore spatial distributed nc files.

Inputs [var is the variable name. If var is 'all' means, then it should]

plot the isofill for all the available variables in the passed path nc or xml file.

path is an absolute nc or xml file path.

outpath is the path to store the images. If it is None means, it should create the least (plotname)directory in the current directory path itself and save it.

region to extract the region from the var data.

if svg is 1, then plot should be saved as svg. if png is 1, then plot should be saved as png.

..note:: isoLevels and isoColors are set inbuilt (some default) range of levels and colors with respect to the variable name of statistical scores.

Written By : Dileep Kumar.R, Arulalan.T

Date : 26.09.2011

`generate_stati_score_spatial_distribution_plots.`**`genSeasonStatiScoreDirs`**(*modelName*, *model-Hour*, *available-Months*)

`genSeasonStatiScoreDirs()`: It should generate the directory hierarichy structure of season statiscore in the plotsgraphspath. And calls the function genIsoFillPlots to make 'isofill' plots and save it inside the appropirate directory, by reading the nc file of the appropirate process season hour statiscore files path.

It should plot for all the vars of that statiscore spatial distributed nc files.

Inputs [modelName is the one of the directories name.] modelHour is the one of the directories name. availableMonths is the dictionary which is generated by fully available months from the timeAxis.

Written By : Arulalan.T

Date : 26.09.2011

3.5 Statistical Scores

3.5.1 Contigency Table & Related Statistical Scores

The module *ctgfunction.py* should helps to calculate the contigency table and its related statistical scores.

For eg : Threat Score, Bias Score, Proabability of detection and more.

`ctgfunction.`**`accuracy`**(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`accuracy()`:Hit Rate ,the most direct and intuitive measure of the

accuracy of categorical forecasts is hit rate. The average correspond- ence between individual nforecasts and the events they predict. Scalar measures of accuracy are meant to summarize,in a single number, the overall quality of a set of forecasts. Can be mislead, since it is heavily influenced by the most common category, usually "no event" in the case of rare weather.

Accuracy= (a+d)/(a+b+c+d); 'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs: Range: 0 to 1

Perfect Score: 1

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS(Page No:236-240)

Links : http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`bias_score`**`(`*obs=None*, *fcst=None*, *th=None*, ***ctg*`)`

`bias_score()`: Bias score(frequency bias)-Measures the correspondence between the average forecast and the average observed value of the predictand. This is different from accuracy, which measures the average correspondence between individual pairs of forecasts and observations.

Bias= (a+b)/(a+c); 'a' -hits, 'b'-false alarm, & 'c'-misses

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs: Range: 0 to infinity

Perfect score: 1

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS(Page No: 241)

Links : http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`contingency_table_2x2`**`(`*obs*, *fcst*, *th*`)`

`contingency_table_2x2()`:Creates the 2x2 contigency table useful for forecast verification. From 2x2 condigency table we can find thse statistical scores such as Hit Rate(HR), Bias(BS), Threat Score(TS), Odds Ratio(ODR)...etc

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

Outputs: A list of values [a, b, c, d]

where a = No of values such that both observed and predicted > threshold

b = No of values such that observed is < threshold and predicted > threshold

c = No of values such that observed is > threshold and predicted < threshold

d = No of values such that both observed and predicted < threshold

Usage:

example: From the contingency table the following statistics can be calculated. >>> HR = (a+d)/(a+b+c+d) >>> ETS = a/(a+b+c) >>> BS = (a+b)/(a+c) >>> ODR = (a*d)/(b*c)

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS

Links: http://www.cawcr.gov.au/projects/verification/#Atger_2001

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.eds` (*obs=None*, *fcst=None*, *th=None*, ***ctg*)

:func:'eds':Extreme dependency score, converges to 2n-1 as event frequency approaches 0, where n is a parameter describing how fast the hit rate converges to zero for rarer events. EDS is independent of bias, so should be presented together with the frequency bias.

EDS={2Log[(a+c)/(a+b+c+d)]/Log(a/(a+b+c+d))}-1; 'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'-correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: -1 to 1, 0 indicate no skill.

Perfect Score: 1

Reference: [1]Stephenson D.B., B. Casati, C.A.T. Ferro and C.A. Wilson, 2008: The extreme dependency score: a non-vanishing measure for forecasts of rare events.

Meteorol. Appl., 15, 41-50.

Link: http://www.cawcr.gov.au/projects/verification.

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.ets` (*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`ets()`**:Equitable threat score (Gilbert skill score), the number of** forecasts of the event correct by chance, 'a_random',is determined by assuming that the forecasts are totally independent of the obs- ervations, and forecast will match the observation only by chance. This is one form of an unskilled forecast, which can be generated by just guessing what will happen.

ETS=(a-a_random)/(a+c+b-a_random) a_random=[(a+c)(a+b)]/(a+b+c+d); 'a' -hits, 'b'-false alarm,

'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: -1/3 to 1, 0 indicate no skill.

Perfect Score: 1

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`far`**`(`*obs=None*, *fcst=None*, *th=None*, ***ctg*`)`

`far()`:False alarm ratio(FAR)-Proportion of forecast events that fail to materialize.

FAR= b/(a+b); 'a' -hits, & 'b'-false alarm

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: 0 to 1

Perfect Score: 0

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS(Page No: 240-241)

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`hss`**`(`*obs=None*, *fcst=None*, *th=None*, ***ctg*`)`

`hss()`:Heidke skill score (Cohen's k), the reference accuracy measure in the Heidke score is the hit rate that would be achieved by random forecasts, subject to the constraint that the marginal distributions of forecasts and observations characterizing the contingency table for the random forecasts, P(Yi) and p(oj), are the same as the marginal distributions in the actual verification data set.

HSS= 2.(a d - bc)/[(a + c)(c + d) + (a + b)(b + d)];

'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: -infinity to 1, 0 indicate no skill.

Perfect Score: 1

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS(Page No: 248-249)

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`kss`**(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`kss()`:Hanssen and Kuipers discriminant(Kuipers Skill Score), the contribution made to the Kuipers score by a correct "no" or "yes" forecast increases as the event is more or less likely, respectively.A drawback of this score is that it tends to converge to the POD for rare events, because the value of "d" becomes very large.

HK= (ad-bc)/[(a + c)(b + d)]; 'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range:-1 to 1 Perfect Score: 1

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS(Page No: 249-250)

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`logodr`**(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`logodr()`: Log Odds ratio, LOR is the logaritham of odds ratio. When the sample is small/moderate it is better to use Log Odds Ratio. It is a good tool for finding associations between variables.

LOR=Log(Odds Ratio); Odds Ratio=ad/bc; 'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: -infinity to infinity, 0 indicate no skill.

Perfect Score: infinity

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.odr`(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`odr()`:Odds ratio, the odds ratio is the ratio of the odds of an event occurring in one group to the odds of it occurring in another group.cThe term is also used to refer to sample-based estimates of this ratio.Do not use if any of the cells in the contingency table are equal to 0. The logarithm of the odds ratio is often used instead of the original value.Used widely in medicine but not yet in meteorology.

OD= ad/bc; 'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: 0 to infinity, 1 indicate no skill

Perfect Score: infinity

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.orss`(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`orss()`: Odds ratio skill score (Yule's Q), this score was proposed long ago as a 'measure of association' by the statistician G. U. Yule (Yule 1900) and is referred to as Yule's Q. It is based entirely on the joint conditional probabilities, and so is not influenced in any way by the marginal totals.

ORSS= (ad-bc)/(ad+bc); 'a' -hits, 'b'-false alarm, 'c'-misses, & 'd'- correct negatives

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: -1 to 1, 0 indicates no skill

Perfect Score: 1

Reference: Stephenson, D.B., 2000: Use of the "odds ratio" for diagnosing forecast skill. Wea. Forecasting, 15, 221-232.

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`pod`**(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`pod()`:Probability of detection(POD), simply the fraction of those occasions when the forecast event occurred on which it was also forecast.

POD= a/(a+c); 'a' -hits, & 'c'-misses

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs: Range: 0 to 1

Perfect Score: 1

Reference: "Statistical Methods in the Atmospheric Sciences", Daniel S Wilks, ACADEMIC PRESS(Page No: 240)

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

`ctgfunction.`**`pofd`**(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

`podf()`:Probability of false detection (false alarm rate), measures the fraction of false alarms given the event did not occur.

POFD=b/(d+b); 'b'-false alarm & 'd'- correct negatives

Inputs: obs - the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass 'ctg_table' kwarg as 2x2 matrix value.

Outputs:

Range: 0 to 1

Perfect Score: 0

Links: http://www.cawcr.gov.au/projects/verification/

`ctgfunction.`**`ts`**(*obs=None*, *fcst=None*, *th=None*, ***ctg*)

:func:'ts': Threat Score (Critical Success Index), a frequently used alternative to the hit rate, particularly when the event to be forecast (as the "yes" event) occurs substantially less frequently than the non-occurrence ("no").

TS=a/(a+b+c); ‘a’ -hits, ‘b’-false alarm, &’c’-misses

Inputs: obs- the observed values has to be a numpy array(or whatever

you decide)

fcst - the forecast values th - the threshold value for which the contingency table needs

to be created (floating point value please!!)

By default obs, fcst, th are None. Instead of passing obs, fcst, and th values, you can pass ‘ctg_table’ kwarg as 2x2 matrix value.

Outputs: Range: 0 to 1

Perfect Score: 1

Reference: “Statistical Methods in the Atmospheric Sciences”, Daniel S Wilks, ACADEMIC PRESS(Page No: 240)

Links: http://www.cawcr.gov.au/projects/verification/

Written by: Dileepkumar R, JRF, IIT Delhi

Date: 24/02/2011

3.6 More

More utilities will be added and optimized in near future.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*