

BASIC ENVISAT ATMOSPHERIC TOOLBOX

Practical on ESA Atmospheric Data Handling

This practical will show you how to access data from various Earth Observation data products by making use of BEAT, the Basic ENVISAT Atmospheric Toolbox, and VISAN, the GUI-based visualization and analysis environment that accompanies it.

Since BEAT and VISAN are powerful tools, we will never be able to cover all their functionality in such a short time. Instead, we will focus our attention on:

- BEAT-II: a set of functions which provide easy-to-use data ingestion and analysis.
- VISAN: a GUI-based analysis and visualization environment based on the Python computer language

Together, they should provide you with a good starting point to use BEAT effectively for your scientific work. When you need functionality that is not provided by BEAT-II, you can later learn how to use the somewhat more complex BEAT-I functions. Also, when you'd rather use MATLAB or IDL for your work, rest assured that the data reading functionality that you will be using in VISAN during this practical is also available for MATLAB and IDL.

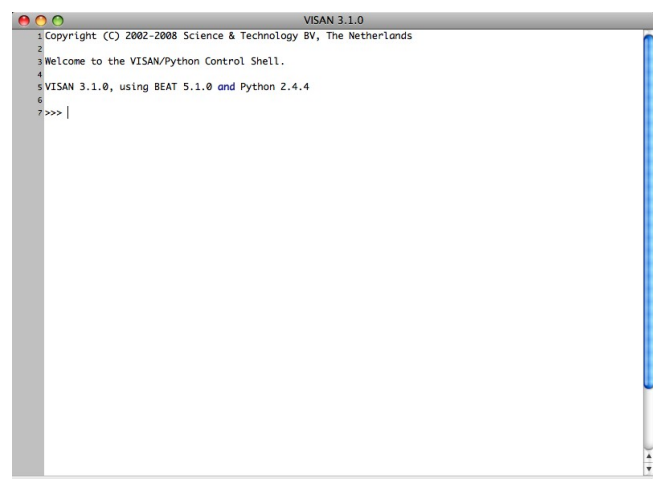
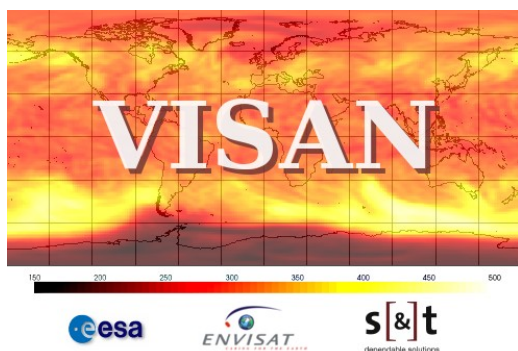
Introduction to VISAN

In this first exercise we will start VISAN and explore some of its features.

Start the VISAN application by clicking on the VISAN icon:



You will (shortly) see the VISAN splash screen, after which you will be provided with the main VISAN window:



```
VISAN 3.1.0
1 Copyright (C) 2002-2008 Science & Technology BV, The Netherlands
2
3 Welcome to the VISAN/Python Control Shell.
4
5 VISAN 3.1.0, using BEAT 5.1.0 and Python 2.4.4
6
7 >>> |
```

In the main VISAN window, you can type commands, which will then be executed, similar to what you would do in MATLAB or IDL. VISAN's built-in language is the popular *Python* scripting language, which has been extended with BEAT specific data types and functions for handling atmospheric data.

The following command shows how you can use the Python language in VISAN as a simple calculator:

```
>>> print sin(pi/3)**2
0.75
```

Python has a nice set of built-in data types, including integers, floating-point numbers, strings, lists, and sets. It also supports large arrays of floating numbers in an efficient way via the *numarray* package, which is available by default in VISAN:

```
>>> t = arange(0, 2*pi, 0.001)
>>> print t
[ 0.00000000e+00,  1.00000000e-03,  2.00000000e-03, ...,
 6.28100000e+00,  6.28200000e+00,  6.28300000e+00]
>>> print len(t)
6284
```

VISAN augments python by including functions to plot 2D and geographic plots, using the `plot()` and `wplot()` functions. It is easy to get help on the many options that these functions provide using the `help()` command:

```
>>> help(plot)
>>> help(wplot)
```

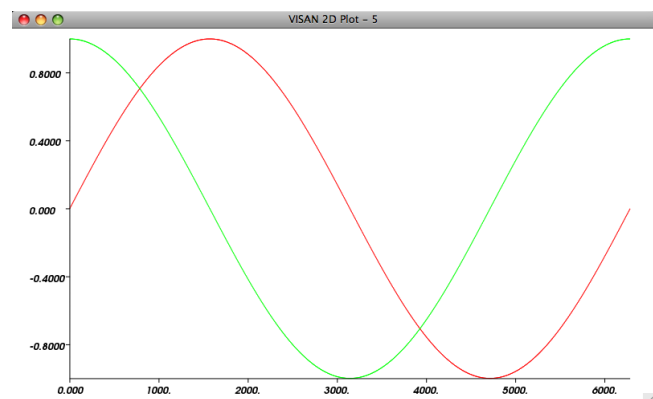
Using the previously defined variable `t` we can make a 2D plot (we will meet geographic world plots in the next exercise):

```
>>> plot(sin(t))
<plotwindow handle>
>>> plot(cos(5*t), sin(3*t))
<plotwindow handle>
```

The 'plot windows' that are created by the commands above are interactive; you can drag in the graph or on the plot axes to pan the graph, and zoom in and out using the right mouse button. This is especially convenient when exploring physical data, where this feature helps you to quickly get the plot that you want.

VISAN allows you to easily create 'overplots', by using the `window` option to the plot command. Each plot command returns a reference to a plot window that can subsequently be used to add a new graph to the plot as follows:

```
>>> plotwin = plot(sin(t))
>>> plot(cos(t), window = plotwin)
<plotwindow handle>
```



Ozone hole breakup

Now that we have explored VISAN a bit, lets have a look at actual atmospheric data.

We will read a series of files containing daily global assimilated ozone data¹ based on measurements from the GOME instrument. The data is from late September 2002, a period that is famous for the ozone hole breakup at the south pole.

Before continuing, you should set VISAN's working directory to the place where the sample data is stored; e.g., if the sample data resides in the directory `/Users/Shared/beat`, execute the following command in VISAN:

```
>>> os.chdir("/Users/Shared/beat")
```

To read the data we will use the BEAT-II interface of BEAT. We only need a single command to read the required data from all available GOME level 4 files:

```
>>> gome14 = beat12.ingest('example1/gome_l4/*')
```

This command has read the data from all the GOME Level 4 files in the `gome_l4` directory and converted the data into a simple 'record' structure:

```
>>> print gome14
           type: 'GOME_FDS_L4'
           time: [30 double]
           grid_latitude: [180 double]
           grid_longitude: [288 double]
           o3_column_grid: [30x180x288 double]
           o3_column_grid_unit: 'DU'
```

Each element in this structure can be freely used/manipulated in the VISAN Python environment.

```
>>> print gome14.grid_latitude[0]
-89.5
```

Note that the BEAT-II ingestion function is also available for IDL and MATLAB², where the function is called `beat12_ingest()` and where you will also get a native IDL `struct` or MATLAB `struct` as result that can be freely manipulated.

The advantage of VISAN is that, unlike IDL and MATLAB, it also provides a way to create a plot from the ingested data using just a single function call. We will call the `wplot()` function with our ingested BEAT-II record and add some parameters to set the color table and range the way we want it:

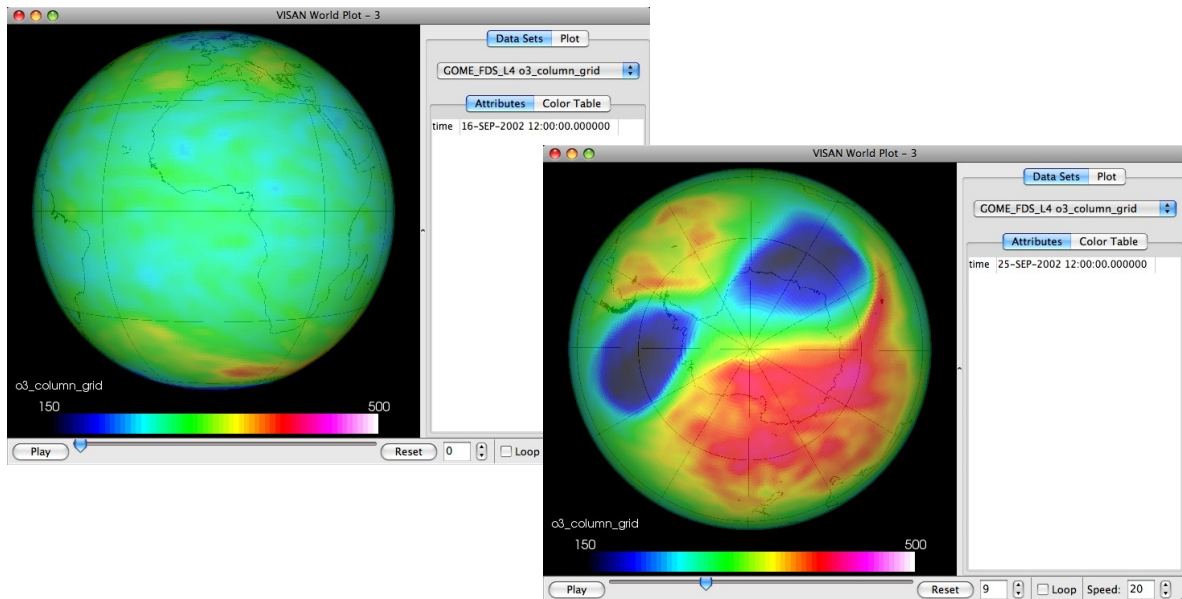
```
>>> wplot(gome14, colortable="Ozone", colorrange=[150,500])
```

Just as with the 2D plot you can drag with the mouse in the plot area to interact with the view. Dragging with the left mouse button pressed will rotate the earth and dragging with the right mouse button pressed will zoom in/out.

1 Provided by the GOME level 4 fast delivery service of KNMI

2 The have this function available, you will have to have the BEAT software package installed and have MATLAB/IDL configured so it will find the BEAT MATLAB/IDL interfaces.

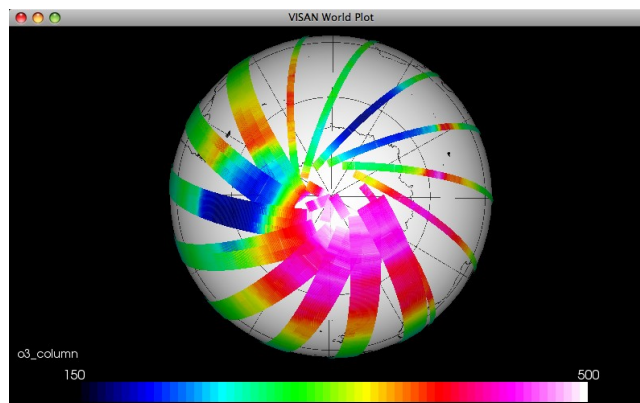
Now rotate the earth such that the south pole is in view and press the play button at the bottom of the window. This will animate through the daily global ozone grids. You will see the ozone hole breakup appear around September 25th.



The GOME level 4 data that you have just plotted provides for each day an assimilation of several GOME orbits covering that day. We will now have a look at what GOME actually measured that day by looking at the GOME level 2 data for 25 September 2002.

To read the data we can use the same `beat12.ingest()` function, but we will now use the first parameter of the function to point to the directory containing GOME level 2 data:

```
>>> gome12 = beat12.ingest('example1/gome_l2/*')
>>> wplot(gome12, colortable="Ozone", colorrange=[150,500])
```



You will see 15 stripes, corresponding with 15 orbits. Note that GOME is only able to measure ozone when it passes the sunlit side of the earth. Also note the difference between the small stripes and the wide stripes, indicating that GOME was measuring in two different modes that specific day.

When you zoom in a bit you will find that the colored pixels show a backward/forward overlapping pattern. This has to do with how GOME performs its measurements. To get a better view of this we can create an overplot of the center latitude/longitude position using the first few GOME ground pixels:

```
>>> w = wplot(gome12.corner_latitude[0:100],gome12.corner_longitude[0:100],
gome12.o3_column[0:100])
>>> wplot(gome12.latitude[0:100], gome12.longitude[0:100], window=w, drawpath=1)
```

If you zoom in on the North Pole region you will see the first part of a GOME orbit. You will see a small startup phase where GOME enters the sunlit side of the earth and where GOME measures with a high integration time. Then GOME starts sweeping from left to right and back again. You will see that the part where the scan goes from left to right (called the forward scan) GOME provides 3 ground pixels, whereas the part from right to left (backward scan) covers only a single pixel.

Since we are often only interested in the part of the scan that provides the 3 ground pixels of the forward scan, the BEAT-II ingestion function provides an option to restrict the ingestion to the forward scan:

```
>>> gome12f = beat12.ingest('example1/gome_l2/*', 'scan_direction=forward')
>>> wplot(gome12f, colortable="Ozone", colorange=[150,500])
```

Now compare this plot with the plot of the GOME level 4 data for September 25th and see if you can detect the same ozone hole split in the GOME level 2 data. Do you see any differences?

Note that there are more options you can provide to an ingestion. An important one is the filter on geolocation. Using `latitude_min`, `latitude_max`, `longitude_min`, and `longitude_max`, you can restrict the region of interest. For instance, to ingest only GOME 2 around the south pole, you could use:

```
>>> gome12sp = beat12.ingest('example1/gome_l2/*', 'scan_direction=forward,
latitude_max=-45')
```

There are often many more options you can provide, including ones that allow you to get different data from a product (e.g. NO₂ columns instead of O₃ columns). For a list of all options for a specific product type have a look at the BEAT-II Data Description documentation. This documentation can be accessed by selecting the VISAN main window and then from the VISAN Help menu choose the 'BEAT-II Data Description' option.

During the September 2002 period the ENVISAT satellite was also operational. Let's try to see what SCIAMACHY measured and compare this with the GOME data. Since SCIAMACHY uses the same forward/backward scanning principle as GOME, we will also use the `scan_direction` filter for the SCIAMACHY level 2 data ingestion. In addition, the SCIAMACHY level 2 products provide their total column values in molecules/cm². In order to better compare them with GOME we will provide the `convert_to_DU` option to the ingestion to have BEAT convert the SCIAMACHY total column values to Dobson Unit values.

```
>>> scia12 = beat12.ingest('example1/scia_l2/*', 'scan_direction=forward,
convert_to_DU')
>>> wplot(scia12, colortable="Ozone", colorange=[150,500])
```

How does SCIAMACHY compare to GOME?

GOME, SCIAMACHY, and OMI

In this next example we will shift the date forward to 25 May 2005 and look at GOME, SCIAMACHY, and OMI for that day.

To make things a bit more interesting, you are provided two different versions of the level 2 products for each instrument. The goal of this exercise is to (visually) compare all 6 data sets, using the techniques you learned in the previous exercise, and get an understanding of the kind of differences that exist. We will provide the ingest commands that you can perform, but you will have to create the plots yourself.

For OMI there are two official level 2 products containing ozone total column data. One is based on the DOAS retrieval method (the OMDOAO3 product) and the other on the TOMS method (the OMT03 product). To ingest the data you can use the following commands:

```
>>> omil2_doas = beatl2.ingest('example2/omi_doas/*')
>>> omil2_toms = beatl2.ingest('example2/omi_toms/*')
```

Since OMI has a very high spatial resolution compared to GOME and SCIAMACHY, the example data set for OMI just contains data for one orbit.

For SCIAMACHY we have provided two data sets of the same level 2 product type, but generated by different versions of the level 2 data processor. One data set was generated by version 2.5 and the other by version 3.01. To ingest the data you can use:

```
>>> scial2_25 = beatl2.ingest('example2/scia_25/*',
'scan_direction=forward,convert_to_DU')
>>> scial2_30 = beatl2.ingest('example2/scia_30/*',
'scan_direction=forward,convert_to_DU')
```

Finally for GOME we have provided the level 2 data sets in both a binary format and in HDF format. To ingest the data you can use:

```
>>> gome12_bin = beatl2.ingest('example2/gome_bin/*','scan_direction=forward')
>>> gome12_hdf = beatl2.ingest('example2/gome_hdf/*','scan_direction=forward')
```

You will see that the GOME data from both data sets is completely identical. The difference between the two data sets is also not their content but how the content is stored. To see this difference we have to go to the BEAT layer 1 level. Try to open both a file from the `example2/gome_bin` directory and one from the `example2/gome_hdf` directory using the VISAN Product Browser. You can open a file in the Product Browser, by selecting the VISAN main window and then, from the menu at the top of the screen, select the 'Browse Product...' option.

The Product Browser will show you the structure of a data product and its contents in the way it is stored on disk. The BEAT-II interface will try to hide all this complexity from you, but with the drawback that it won't give you access to all data that may be available in the product. If you want to read information from a file that is not accessible via BEAT-II, you can still use the BEAT-I interface (using `beat.open()`, `beat.fetch()`, and `beat.close()`), which is not covered by this practical, to get access to that information. However, this will require you to get familiar with how the product is structured (as shown in the Product Browser).

Additional examples

If you still have time left there are a few additional data sets that you may have a look at.

GOME level 1 data

GOME Level 1 spectra for a single orbit of GOME for 25 September 2002:

```
>>> gome11 = beat12.ingest('moredata/gome_11/*')
```

You can plot the spectra with `plot()` and show the measurement locations with `wplot()`.

GOMOS level 1 data

A single GOMOS transmission spectra file for 7 September 2008:

```
>>> gomos12 = beat12.ingest('moredata/gomos_11/*')
```

You can plot the spectra with `plot()` and show the locations of the profile points with `wplot()`.

GOMOS level 2 data

All GOMOS ozone profiles for 7 September 2008:

```
>>> gomos12 = beat12.ingest('moredata/gomos_12/*')
```

You can plot the profiles with `plot()` and show the locations of the profile points with `wplot()`.

MIPAS level 2 data

All MIPAS ozone profiles for 25 September 2002:

```
>>> mipas12 = beat12.ingest('moredata/mipas_12/*')
```

You can plot the profiles with `plot()` and show the locations of the profile points with `wplot()`.

You can also try getting different data from data products that you already used:

SCIAMACHY NO2 total column data

```
>>> scial2_no2 = beat12.ingest('example1/scia_12/*', 'data=nadir_uv_1')
```

SCIAMACHY O3 profiles

```
>>> scial2_o3p = beat12.ingest('example1/scia_12/*', 'data=limb_uv_0')
```

GOME NO2 total column data

```
>>> gome12_no2 = beat12.ingest('example1/gome_12/*', 'include=no2_column')
```