Modelling and Forecasting Facility internal report

Python tools for BRIFS operational verification with in-situ observations

Matjaz Licer

May 13, 2016 [updated June 10, 2016]

Contents

1 Objective 2

2 The required environment 2

3 A walkthrough through the `performBRIFSverification.py` 2

4 An example: event 20160331 8

1 Objective

This is documentation for BRIFS system verification Python tools. These tools allow us to operationaly (daily) compare BRIFS system outputs with all available in situ data from SOCIB observational network and plot high-quality images of comparisons. (Adapting the tool to be used for operational WMOP verification should also be straightforward.)

The verifications were tested at SOCIB SCBD067 computer (at IMEDEA) from directory:

```
/home/mlicer/BRIFSverif/pyVerif/sourceCodes
```

for several dates from the `/home/rissaga/new_setup/Archive/` folder. The required codes for the verification run are the following (see also Figure 1):

- `performBRIFSverification.py`: the `main()` script of the verification
- `getAllObservations.py`: retreives available netCDF filenames with relevant observations for the required date using SOCIB `DataDiscovery` utility.
- `obsData.py`: reads the required observational data from the relevant netCDF files.
- `modelData.py`: reads ROMS and WRF model outputs and extracts point values at the locations of available observational stations.
- `mergeWRF.py`: merges in time extracted WRF points for three consecutive days of each individual BRIFS run.
- `basicStatistics.py`: loops over each station, resamples the observed and modeled data to 1-minute temporal resolution, gets rid of NaNs (these steps are performed using `Pandas` library) and computes basic statistical scores of the forecast.
- `plotBRIFS.py`: plots the comparisons.

2 The required environment

The system needs the following environment components/libraries to run:

- System: HDF5 and netCDF C libraries, built with shared and OpenDAP support.
- Python: `numpy, scipy, matplotlib, netCDF4`.

3 A walkthrough through the `performBRIFSverification.py`

Firstly: the codes themselves are thoroughly commented and these comments are probably enough to understand the workflow (see also Figure 1). Nevertheless we walk through the main

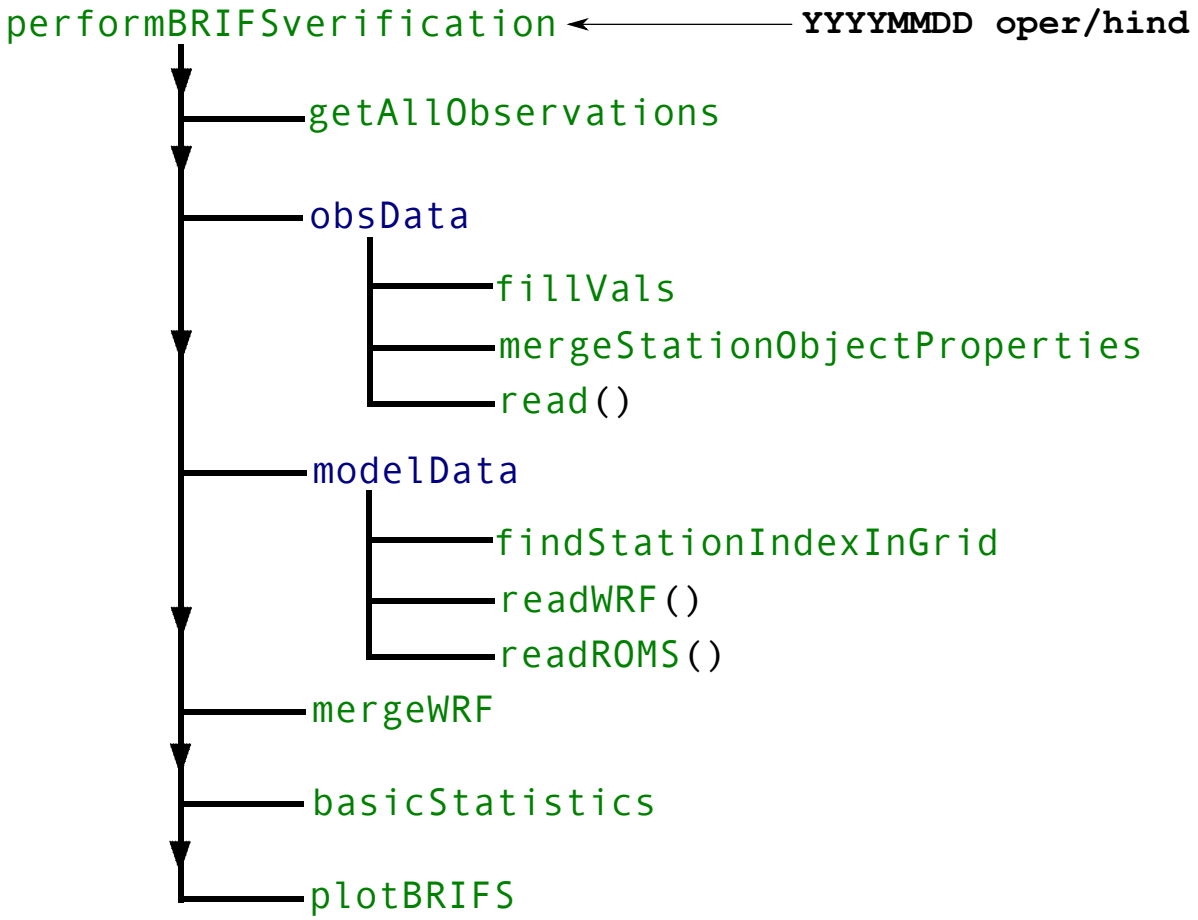

FIGURE 1: Code structure of the verification tool. Green names refer to functions/methods, blue names refer to objects.

code `performBRIFSverification.py` to explain its flow. It is run with, for example:

```
./performBRIFSverification 20150611 oper
```

or

```
./performBRIFSverification 20150611 hind
```

As you can see, it takes two input parameters:

- `strdate`: which is YYYYMMDD date string of the date we want to perform the verification on.
- `oper/hind`: string indicating whether we read operational results (WRF forced with GFS) or hindcast results (WRF forced with FNL).

An output directory for plots with the YYYYMMDD name is created in the

`/home/mlicer/BRIFSverif/pyVerif/YYYYMMDD_oper/hind/`

directory. The code begins by hard-coding the directories of the WRF and ROMS outputs, and a directory for saving plots:

```
# set wrf and roms netCDF output directories:
wrfdir = '/home/rissaga/new_setup/Archive/Outputs/WRF/'+strdate+'/'
romsdir = '/home/rissaga/new_setup/Archive/Outputs/ROMS/'
plotdir = '/home/mlicer/BRIFSverif/pyVerif/'+strdate+'/'
```

These are the only parameters that should ideally be changed by hand. All the rest, like `timeWindow = 48` etc. should be kept as is - at least the codes were not tested for other values, and BRIFS itself does not contain data beyond 48 hours anyway.

Below are arrays with names of the fields that we want to read from the observation netCDF files (these fields need to be present in the netCDFs in exactly the same form):

```
# specify fields for comparisons:

 observationFields=['time','LON','LAT',
'SLEV','QC_SLEV',
'WTR_PRE','QC_WTR_PRE',
'AIR_PRE','QC_AIR_PRE']
```

The `wrfFields` list determines which fields need to be computed and/or extracted from the WRF netCDF at each station location:

```
 wrfFields=['location','Times','XLONG','XLAT', 'pointLon','pointLat',
'pointMSLP']
```

Note that in the line `wrf_array = modelData(...).readWRF()` the *entire* WRF netCDF gets read in order to compute the mean sea level pressure from the surface pressure. As noted above, the `wrfFields` list merely determines what will be *output* to the `wrf_array` from the reading subroutine `modelData.readWRF()`. In other words: the output from `wrf_array = modelData(...).readWRF()` is an array which contains, for every observation station, one Python data object with the following fields (if `DataDiscovery` returns 7 stations, then `wrf_array` will be an array containing 7 objects with the following fields):

- `location`: the string name of the location of the sensor
- `Times`: the datestring vector of times in the WRF netCDF
- `XLONG, XLAT`: 2D grid of the WRF domain
- `pointLon, pointLat`: longitude and latitute of the sensor location
- `pointMSLP`: a `Times`-long mean sea level pressure timeseries from WRF at the location of the sensor location.

It is exactly the same with ROMS results, only the model depth `h` at the station location is added to the object containing data for particular station, and `pointSSH` is the `ocean_time`-long timeseries of ROMS sea surface heights at individual station location. The call to `roms_array = modelData(...).readROMS()` will contain for each station one object with the following fields:

```
romsFields=['location','ocean_time','lon_rho','lat_rho','h','pointLon','pointLat',
'pointSSH']
```

Let's proceed with the code `performBRIFSverification.py`: after determining the contents of all the lists mentioned above, the code then

1. reads the observational netCDF files to `stations` data object and
2. determines all the sensors types (`sensorType` array) at respective locations. It then merges files (and sensor types) from different months(years) if neccessary.

Both tasks are performed with the call:

```
stations,sensorType = obsData(fileList,sensorType,observationFields).read()
```

A `stations` array is an array with the size of the number of available stations from `DataDiscovery`. It contains that many objects with observation fields `observationFields` from each particular station.

After reading the observations we extract WRF and ROMS model results at stations locations, as explained in the paragraphs above. WRF needs to be extracted for all three days of a particular BRIFS run, so:

```
 # extract WRF for available grid points:
wrf_yesterday =
modelData(stations,yesterday,wrfFields,romsFields,wrfdir,romsdir).readWRF()
wrf_today =
modelData(stations,today,wrfFields,romsFields,wrfdir,romsdir).readWRF()
wrf_tomorrow =
modelData(stations,tomorrow,wrfFields,romsFields,wrfdir,romsdir).readWRF()
```

The call to

```
 # extract ROMS
roms =
modelData(stations,startdatenum,wrfFields,romsFields,wrfdir,romsdir).readROMS()
```

reads the ROMS files and extracts SSH at stations locations, as explained above.

The call to

```
    wrf_t_3days,wrf_p_3days =
mergeWRF(stations,wrf_yesterday,wrf_today,wrf_tomorrow)
```

simply merges extracted points from three consecutive days to one array of objects, containing all three days of data.

The call to

```
    # compute basic statistics (BIAS, RMSE, CORR):
stats = basicStatistics(strdate,sensorType,stations,wrf_t_3days,wrf_p_3days,roms)
```

computes basic statistical estimators of the forecast. Specifically, $BIAS$, $RMSE$ and correlations (only for air pressures) are computed from model (m) and observation (o) timeseries. As usual, for any timeseries (with N_t records) of point variable p (with standard deviation σ) at station s, $BIAS$ is computed as

$$BIAS(m,o) = {N_t}^{-1} \sum_{k=1}^{N_t} (p_m^s(k) - p_o^s(k)) \,, \tag{1}$$

$RMSE$ is computed as

$$RMSE(m,o) = \sqrt{{N_t}^{-1} \sum_{k=1}^{N_t} (p_m^s(k) - p_o^s(k))^2} \,, \tag{2}$$

and the correlation coefficient $\rho_{m,o}$ is computed following Pearson as

$$\rho_{m,o} = \frac{COV(m,o)}{\sigma_m \sigma_o} \,, \tag{3}$$

where $COV(m,o)$ is the covariance of both timeseries

$$COV(m,o) = {N_t}^{-1} \sum_{k=1}^{N_t} (p_m^s(k) - \overline{p_m^s})(p_o^s(k) - \overline{p_o^s}) \,. \tag{4}$$

The results of this call is *stats* numpy array, which contains the following parameters for every station:

- `AIR_PRE_BIAS`: air pressure bias between model and barometer observations
- `AIR_PRE_RMSE`: air pressure root mean squared error between model and barometer observations
- `AIR_PRE_CORR`: air pressure Pearson correlation between model and barometer observations
- `WTR_PRE_BIAS`: water pressure bias, computed in meters, between model and AWAC observations
- `WTR_PRE_RMSE`: water pressure root mean squared error, computed in meters, between model and AWAC observations

- `SLEV_BIAS`: sea level bias between model and tide-gauge observations
- `SLEV_RMSE`: sea level root mean squared error between model and tide-gauge observations

The call to

```
 # plot graphs:
plotBRIFS(strdate,sensorType,stations,wrf_t_3days,wrf_p_3days,roms)
```

plots the relevant graphs. Currently only parameters `SLEV, AIR_PRE` and `WTR_PRE` are taken into account. The pictures will be output to the YYYYMMDD `plotdir` specified in the `performBRIFSverification.py` code.

One last thing to note is that the `plotBRIFS.py` function also performs high-pass filtering of SSH signals prior to comparisons. This is done because observations reflect tides, surges and other SSH variability that is not included in the BRIFS ROMS forcing (which is forced by the atmospheric pressure alone). We therefore filter out all the low frequency variability and plot the high frequency part of the SSH signal. Currently we use a 3rd order ($n = 3$) Butterworth high-pass filter with a transfer function

$$G^2(\omega) \propto \frac{1}{1 + \left(\frac{\omega}{\omega_c}\right)^{2n}} \tag{5}$$

where the low frequency cut-off was set to $\omega_c = 25$ minutes. In general this means that the components with frequencies below cut-off will have $-3\,\mathrm{dB} \sim 10^3$ times less amplitude than components at lower frequencies. The high-frequency cut-off was set to 3.75 seconds, which is effectively infinitely high, since it's above the observational or model's sampling frequency. Keep in mind that the filtering function within the `plotBRIFS.py`, namely `butter_bandpass_filter(data, lowcut, highcut, fs, order)` takes as input `lowcut` and `highcut` - these are the low and high frequency cut-offs in terms of the Nyquist rate $N_q = F_s/2$ where F_s is the sampling frequency (usualy of the order of 1/60. s^{-1}). Setting `lowcut = 0.02`, as we have, therefore implies a 25 minute cut-off period:

$$\frac{0.02}{N_q} = \frac{0.04}{F_s} = \frac{1}{25\,\mathrm{min}} \tag{6}$$

Similarly, setting `highcut = 8`, as we have, therefore implies a 3.75 second cut-off period:

$$\frac{8}{N_q} = \frac{16}{F_s} = \frac{1}{3.75\,\mathrm{seconds}} \tag{7}$$

By the way: if for some reason Butterworth filter starts to issue warnings about ill-conditioned coefficients and meaningles results, it may help to lower the order of the filter to $n = 2$. The higher the filter the cleaner the cut-off - but also the filter becomes more unstable. In our case $n = 2$ would also completely suffice for general purposes.

4 An example: event 20160331

By executing the call:

```
./performBRIFSverification 20160331
```

the following observations are found by the `DataDiscovery` utility and data is extracted from WRF and ROMS for their respective locations:

```
calamillor :  201603010000 201604302359 scb-met001
playadepalma :  201603080920 201604302359 scb-met012
sonbou :  201603010001 201604302359 scb-met011
ciutadella :  201603010000 201604302359 ime-awac001
lamola :  201603010000 201604302359 scb-baro002
lamola :  201603010000 201604302350 scb-sbe37001
lamola :  201603010000 201603101940 scb-sbe37002
bahiadepalma :  201603211000 201604302350 scb-met008
canaldeibiza :  201603151520 201604302350 scb-met010
andratx :  201603010000 201604302359
coloniasantpere :  201603010000 201604302359
coloniasantpere :  201603010000 201604302359 pib-baro005
coloniasantpere :  201603010000 201604302359 pib-sbe54004
pollensa :  201603010000 201604302359
pollensa :  201603010000 201604302359 scb-baro001
pollensa :  201603231101 201604302359 scb-sbe26002
portocristo :  201603012352 201604302359
portocristo :  201603012352 201604302359 pib-baro003
portocristo :  201603012352 201604302359 pib-sbe54006
sarapita :  201603010000 201604302359
sarapita :  201603010000 201604302359 scb-sbe26003
santantoni :  201603010000 201604302359
santantoni :  201603010000 201604302359 pib-baro004
santantoni :  201603010000 201604302359 scb-wlog001
esporles :  201603010000 201604302359 scb-met003
parcbit :  201603010000 201604302359 scb-met004
ciutadella :  201603010000 201604302359 scb-baro005
```

The plots are generated if the QC flags indicate that there are non-NaN values in the data. In this case the Figures 2 - 16 on the following pages were generated.

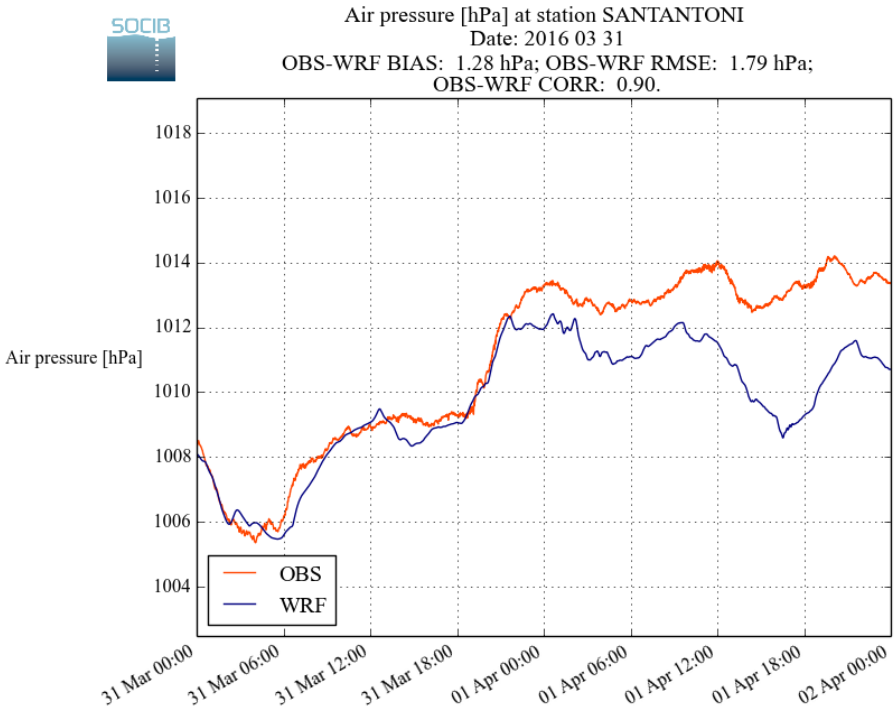

FIGURE 2

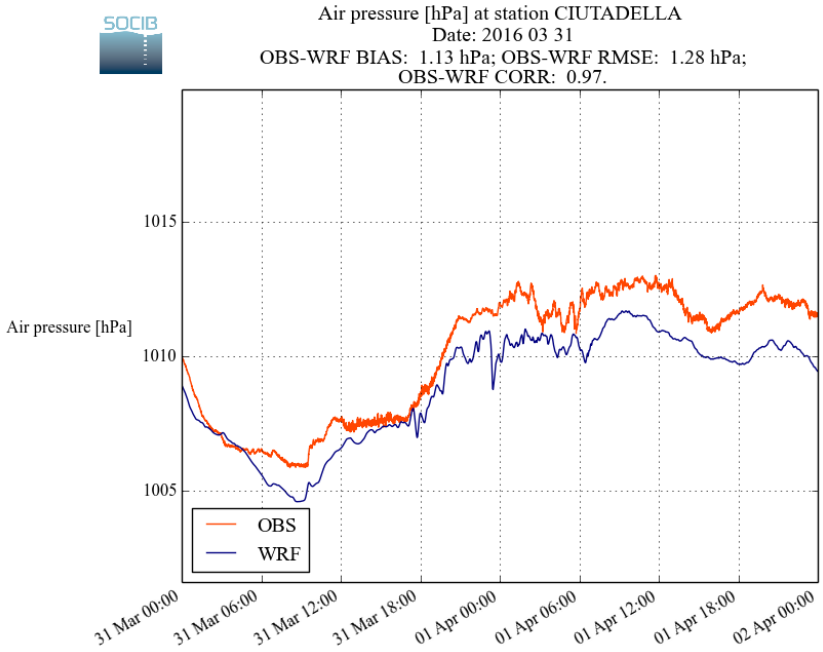

FIGURE 3

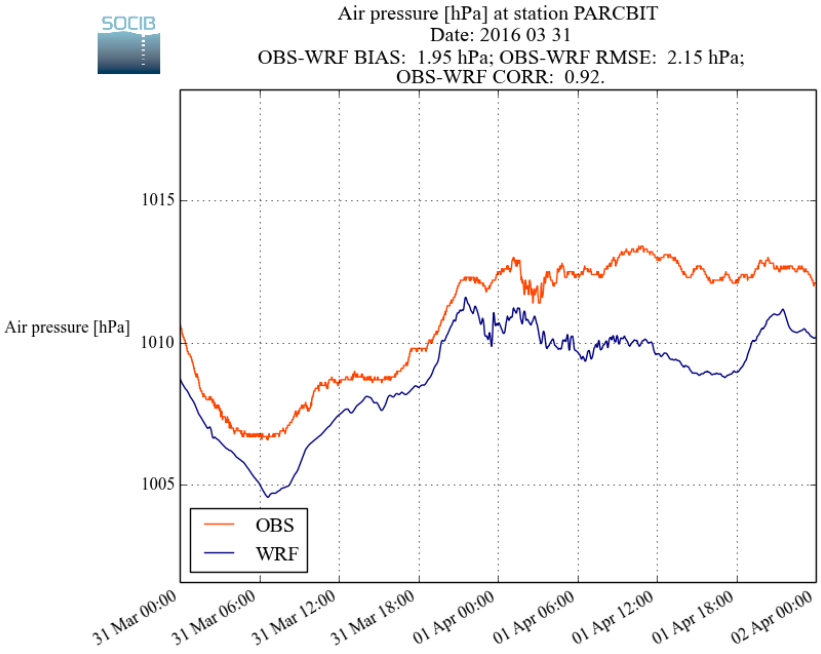

FIGURE 4

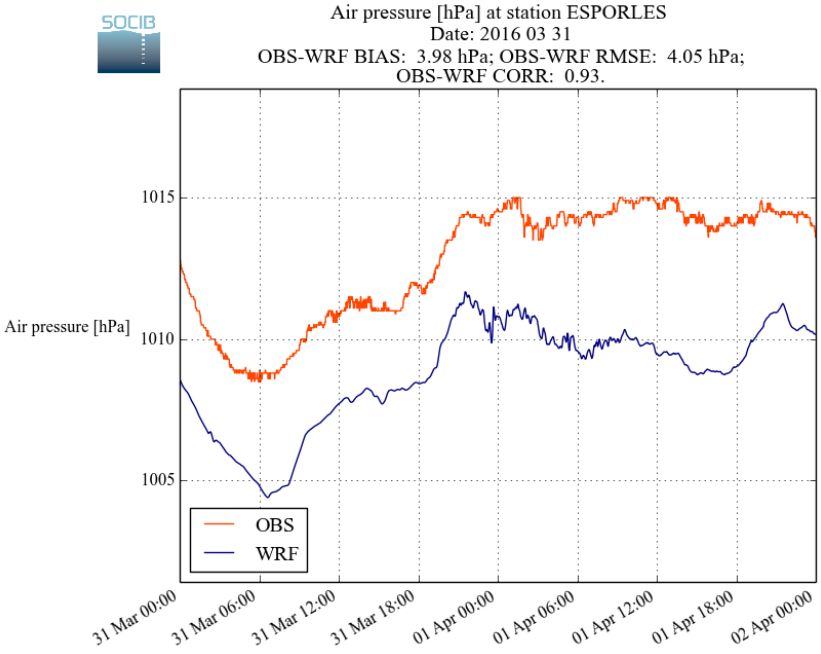

FIGURE 5

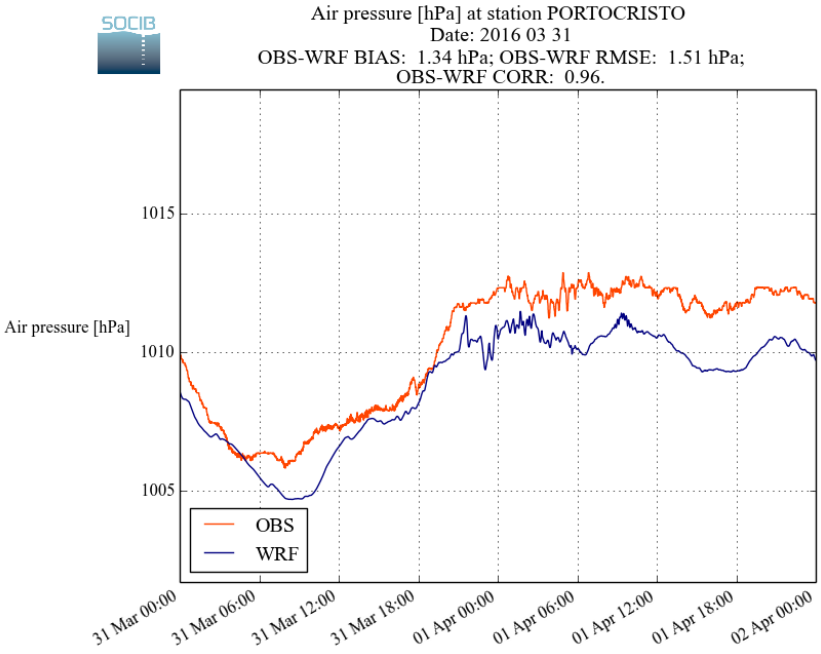

FIGURE 6

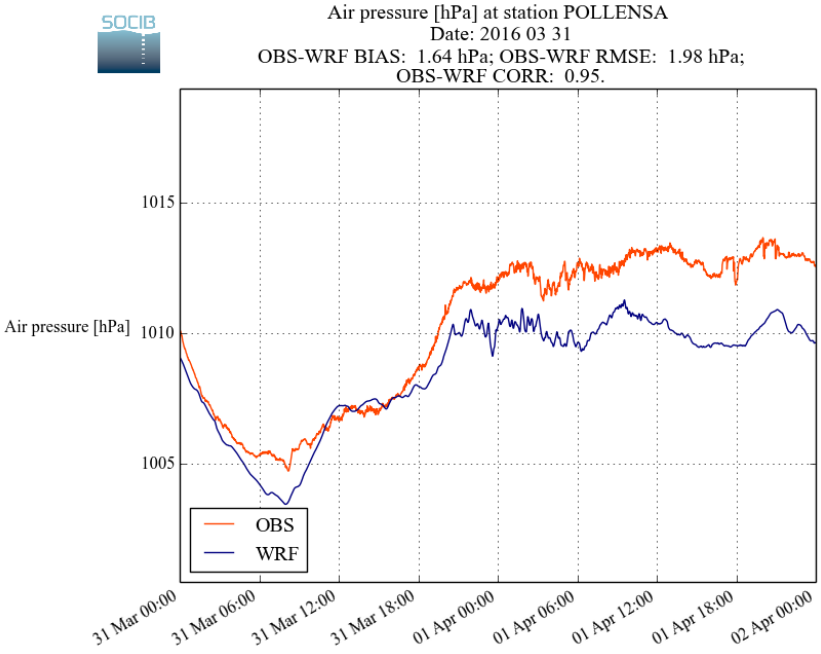

FIGURE 7

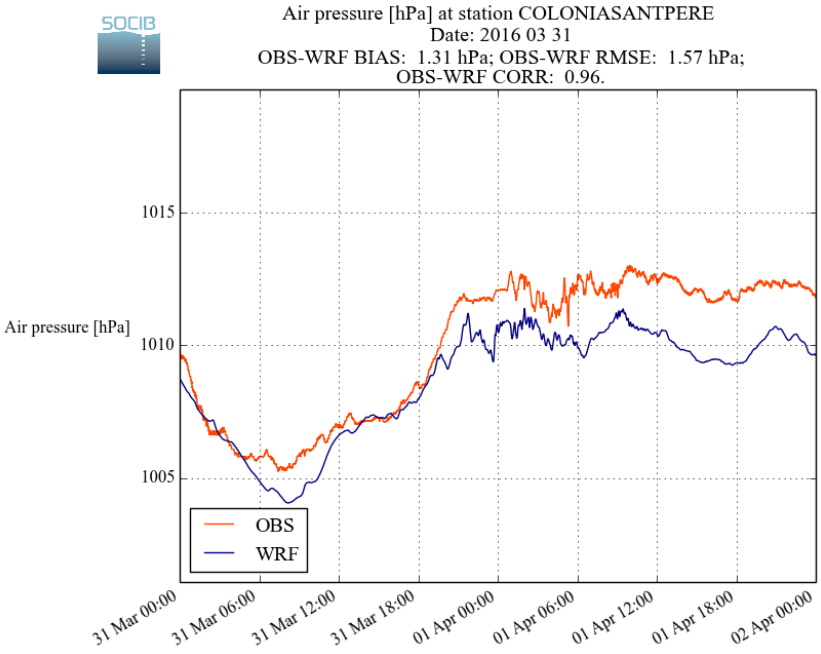

FIGURE 8

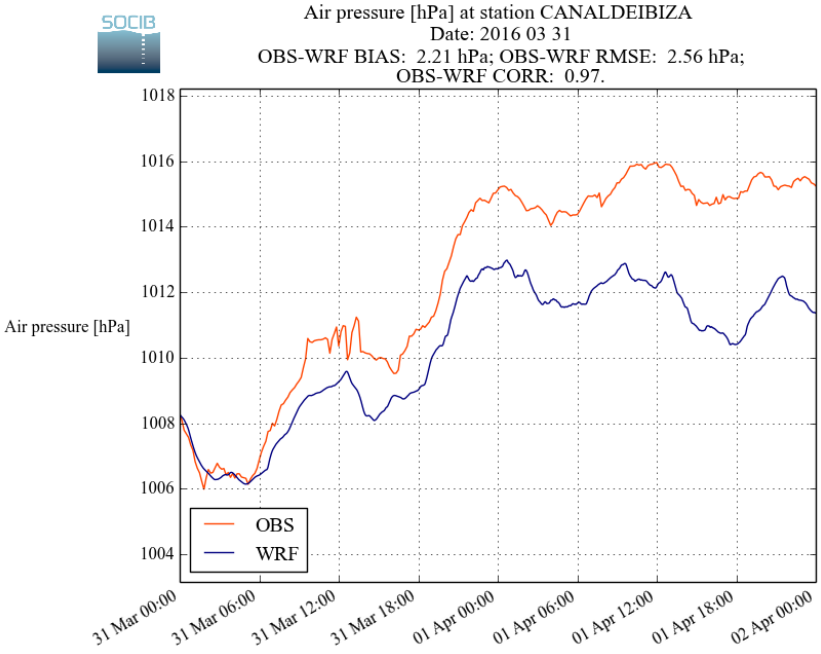

FIGURE 9

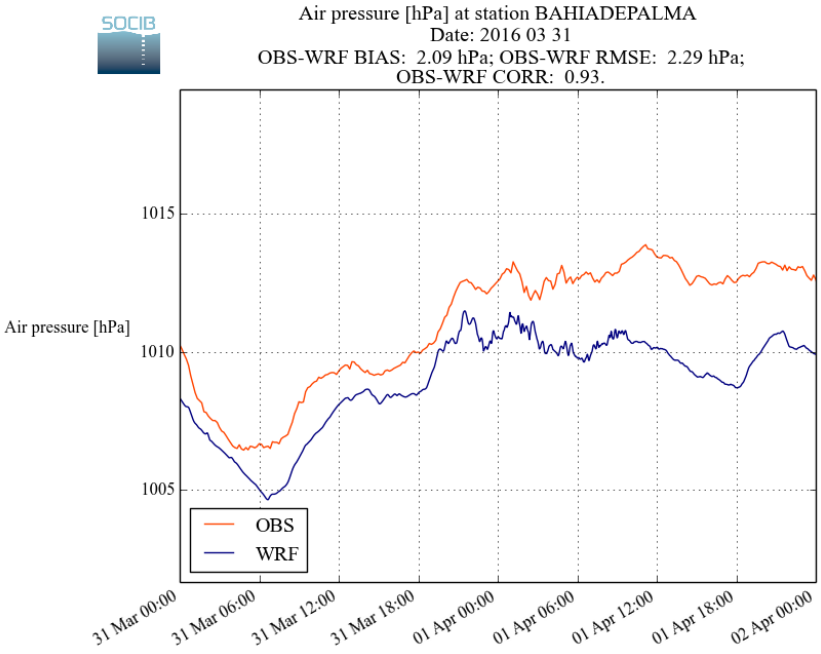

FIGURE 10

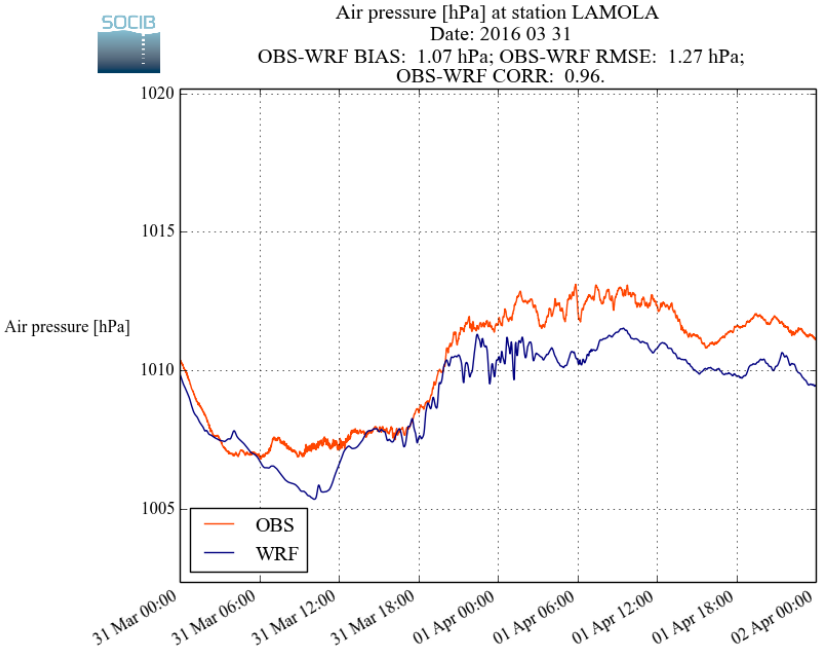

FIGURE 11

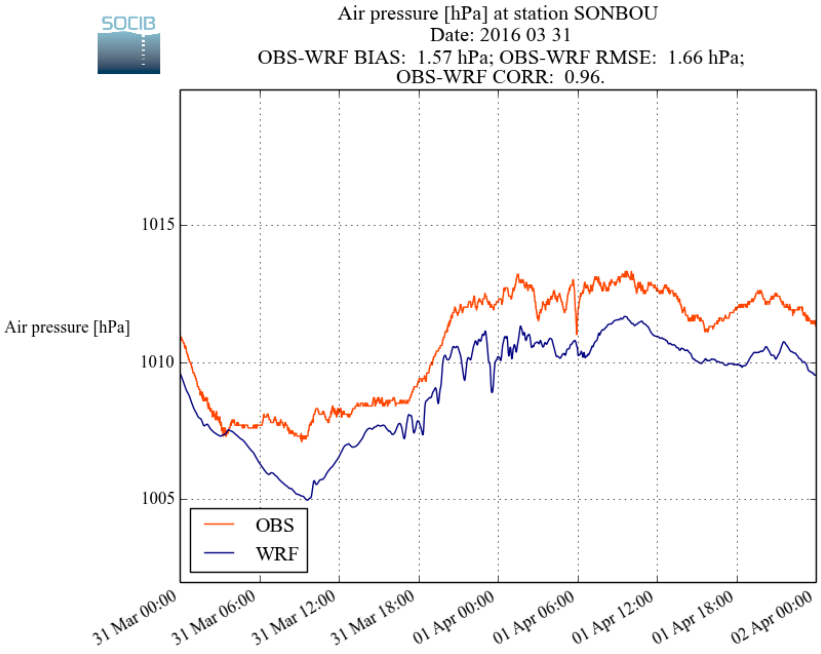

FIGURE 12

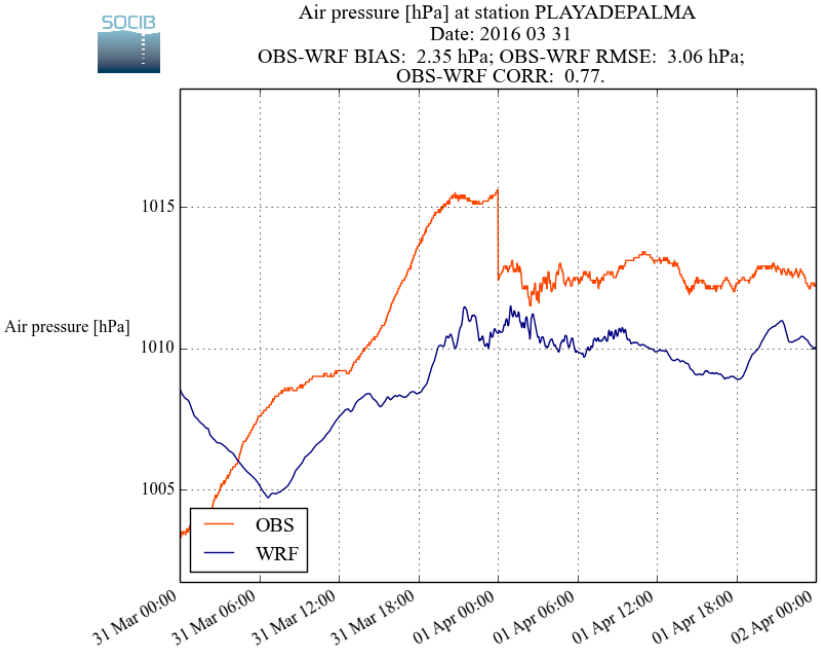

FIGURE 13

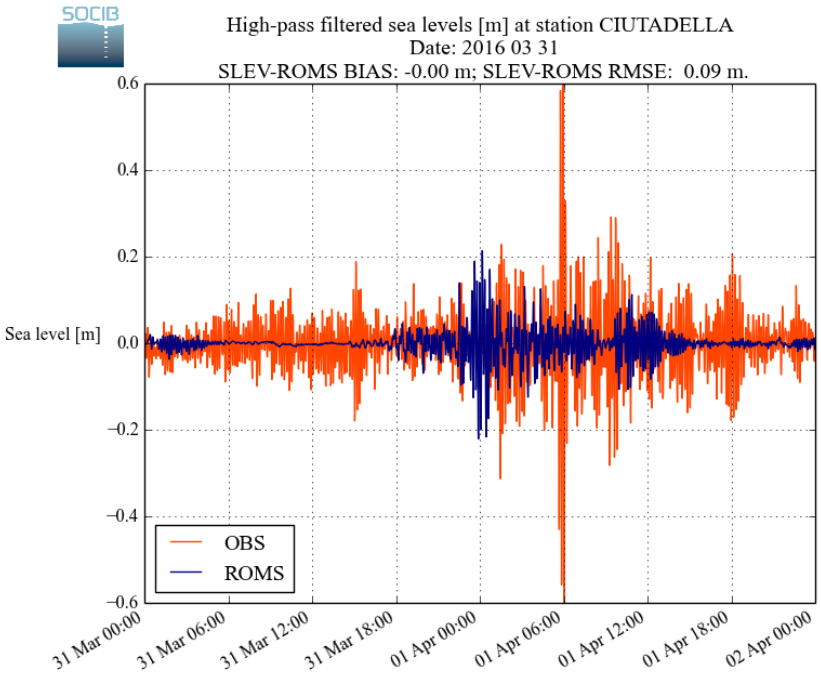

FIGURE 14

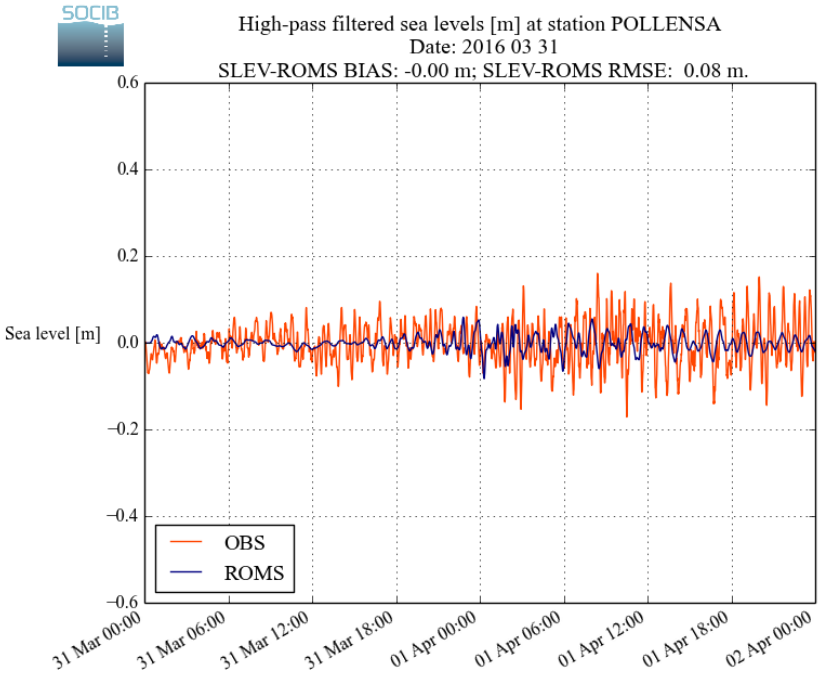

FIGURE 15

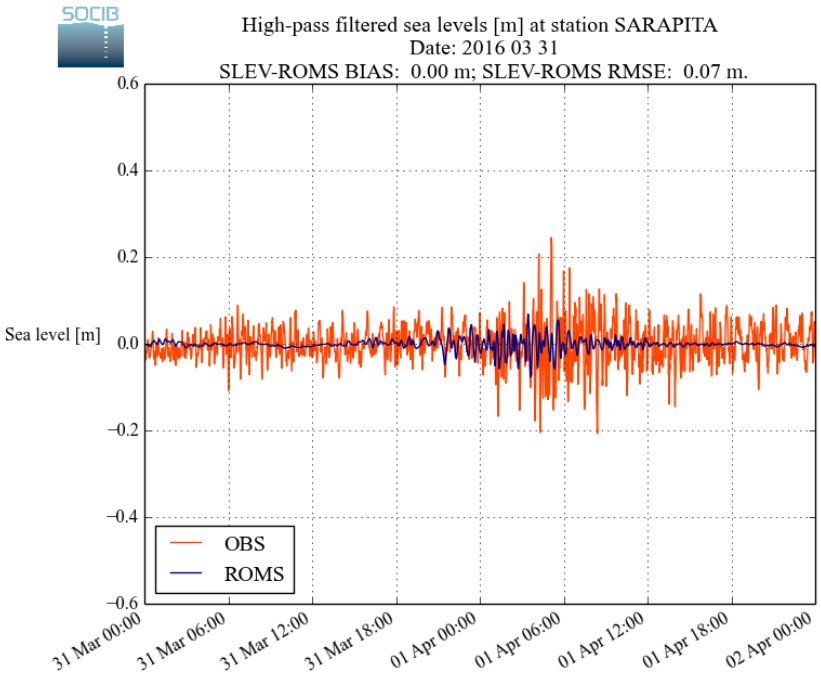
SOCIB
High-pass filtered sea levels [m] at station SARAPITA
Date: 2016 03 31
SLEV-ROMS BIAS: 0.00 m; SLEV-ROMS RMSE: 0.07 m.
Sea level [m]
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
OBS
ROMS
31 Mar 00:00
31 Mar 06:00
31 Mar 12:00
31 Mar 18:00
01 Apr 00:00
01 Apr 06:00
01 Apr 12:00
01 Apr 18:00
02 Apr 00:00

FIGURE 16