Berkeley **H**igh **R**esolution (**BEHR**) Retrieval: Readme

Josh Laughner

May 12, 2016

Summary:

The Berkeley High Resolution Retrieval is a high resolution NO_2 retrieval based on the NASA OMNO2 product from the Ozone Monitoring Instrument (OMI) aboard the Aura satellite. This retrieval improves the standard OMNO2 product in several ways:

1. A higher resolution terrain product (GLOBE Terrain Database) is used to calculate the terrain pressure of the pixels

2. A more frequent and finer resolution albedo is used, taken from the Moderate Resolution Imaging Spectrometer (MODIS) instruments aboard the Terra and Aqua satellites.

3. A MODIS cloud fraction is available to use in rejecting cloudy pixels, as the OMNO2 cloud fraction tends to overestimate cloud fractions if highly reflective surfaces are present

This document will describe the current state of BEHR, including its file structure and a changelog. As of this writing, the code is maintained in a Git repository on my machine. I will try to keep the main matter of this document up-to-date; however, if there is a discrepancy between the changelog and the main matter, defer to the changelog.

Contents

1 Authors

BEHR was initiated by Ashley Russell, who completed her Ph. D. in 2012. She showed that using high-resolution albedo and terrain data improved the OMI NASA Standard Product retrieval (see her papers in the Literature section below). Check the group website for her current contact information.

Luke Valin also completed his Ph. D. in 2012; he helped Ashley run the WRF-Chem simulations needed to get the high-resolution NO_2 profiles.

Josh Laughner took over development in 2013.

If you contribute to the development of BEHR, add your name and contribution to this list and update the change log.

2 Literature

3 Getting started

3.1 Before you begin

You'll want to make sure you have the following software or utilities installed:

- **All platforms:**
  - MATLAB
  - Git version control system
  - Python (optional but recommended)
- **Windows:**
  - An SSH client like puTTy
  - Cygwin - a bash shell for Windows. Includes capability to make SSH connections.
  - wget - a method for retrieving remote files, install by selecting it as a package during Cygwin setup
- **Mac:**
  - Find the "Terminal" app, this is where you can SSH from and use commands like `wget`
- **Other:**
  - Get an account on the computing cluster. Currently this is the Savio cluster (`http://research-it.berkeley.edu/services/high-performance-computing/`). Generally the process consists of emailing the cluster support and having Ron approve your request.

MATLAB is used to run BEHR itself. Python is another language that some utilities for BEHR are written in currently. None are essential. Bash is a shell—basically a text based OS interface—that Unix based systems use. It's the easiest way to download large amounts of satellite data with the command line utility `wget`, and is used in much of the BEHR automation and in the development branch. SSH is a secure protocol that allows remote, command-line access to machines set up to accept SSH connection (this includes the file server, Savio cluster, and satellite download computer).

Some resources if you are new to any of these languages:

- MATLAB:

- Git:
- Python:
- Bash:
- SSH:

3.2 Quickstart

1. Clone the BEHR_GitRepo, MiscUtils, and BEHR_MatlabClasses_GitRepo repositories (Table 1, pg. 7) and add them to your Matlab path.

2. Mount the share-sat share on the Synology NAS at 128.32.208.13 (you'll need the login credentials from another group member).

3. Run BEHR_path_setup.m under BEHR/Utils/BEHR_path_setup.m to create the function BEHR_paths.m that will point to various directories on your computer and the file server that BEHR needs.

4. Try running read_omnno2_v_aug2012.m for one or two days. Make sure it saves to some temporary directory and *not* the main OMI_SP directory on the file server.

5. Do the same for BEHR_main.m, also saving to a temporary folder.

4 Retrieving NASA data

4.1 Group server file locations

A shared file server has been dedicated to storing satellite data locally. It is located at the IP address 128.32.208.13. To connect to this server, your computer will need to have a UC Berkeley IP address. Connecting through an ethernet wall port is the recommended method, but you can also access it by connecting to the RC-Lab wifi network or from anywhere as long as you are connected to the Berkeley VPN.

If you are working with satellite data, you will want to "mount" the file server as a network share on your computer; this will let your computer treat the files as being on an external hard drive connected to it, meaning you will not need to copy the files to your computer in order to access them. Instructions to connect are included in the PDF on the shared group Google drive folder; the current safety officer should know where that is.

All satellite data is stored on the share-sat shared folder, in the SAT subfolder. It is organized first by instrument (OMI, MODIS) then product (OMNO2, MYD06_L2, etc). Products necessary to run BEHR are organized chronologically in subfolders by year (and then month in the case of OMNO2). Other products that BEHR does not rely on are sorted in various ways.

4.2 Downloading from web sites

- **OMNO2:**

  NASA OMNO2 data (the NASA OMI NO_2 product) can be downloaded at `http://mirador.gsfc.nasa.gov`. Search for the keyword "omno2" and specify your date range. We usually leave the location empty. This will return several products; BEHR uses the "1 orbit L2 swath" version. Click on "View files," then on the following page "Add all files in all pages to cart." We don't need any special options, so choose "Continue to cart" and then on the next page "Checkout." On the next page choose "URL List (Data)," this will open in a new tab or window. Save these URLs in a plain text file in the `download_staging` folder under OMNO2 on the file server.

  To download, navigate to the `download_staging` folder and run `wget -i "<list file>"` where `<list file>` is the file you just saved. Keep the window open until this finishes. To sort these into the proper folders, go up one directory and into the `scripts` folder. Run `sortscript.sh` by typing `./sortscript.sh`. All the OMNO2 files will be moved into the appropriate year/month folders.

- **MODIS data:**

4.3 Automated downloading

The necessary satellite data for BEHR is being downloaded automatically by the black Compaq computer in room B47. Each week, this computer queries the relevant archives for new files, compares the remote and local file lists, and retrieves any new files from the remote archive. It will not intelligently acquire new versions, so it will need updated accordingly as new product versions are released, or additional satellite products are required by BEHR.

It is also running `read_omno2_v_aug2012.m`, `BEHR_main.m`, and `BEHR_publishing_v2.m` each week to produce new BEHR files. This is keeping the website (§8) as up-to-date as possible.

Here the downloading process itself will be described. All scripts can be found in the `Downloading` folder of the BEHR git repository (see §5). For information on how the computer and its OS were configured for this, see Appendix B.

- **OMNO2:**

  Automatic OMNO2 downloads required a data subscription with GISC Mirador (the current one was setup by Josh Laughner in his name). The process consists of two steps:

  1. `order_omno2.sh` - this script uses the subscription to request OMNO2 files from the last 30 days. It then waits for the Delivery Notice to be send by Mirador to the *nasa* user set up on this computer specifically to receive these notices by SFTP (see Appendix B), then copies these to one folder as a record and to the `download_staging` folder on the file server.
  2. `get_omno2.sh` - this script waits for `order_omno2.sh` to copy the delivery notice to the `download_staging` folder, then downloads each non-metadata file that does

not exist locally (using wget). As it downloads each file, it sorts it into the proper year/month folder.

- **MODIS:**

  Currently, albedo and cloud files are handled differently; this may change. `get_modis.sh` searches for both sets of files for the last 90 days. (The longer time frame is because the albedo is a combined Aqua/Terra product, averaged over 16 days, and so is not released in real time.) In both cases, it looks for remote files not present locally. For the albedo, it does so by using a feature of wget which can retrieve remote directory listings. It checks the listings on the LADSWEB FTP server against local files, and then retrieves those files not present locally (also using wget).

  Cloud files were more involved, because these become very large as many of them are downloaded. Since we look at individual MODIS granules for clouds (instead of the global gridded product for albedo) we want to restrict the cloud files to the lat/lon boundaries of the US (or other region of interest), plus a 5-10° buffer to ensure all granules are retrieved.

  To do this, we use the *Simple Object Access Protocol* through the Python module SOAPpy to send a request for all MYD06_L2 files in the given time and space range, using the Python script `automodis.py`. This places a list of URLs into a file that `get_modis.sh` can then check against local files.

- **Reading the files:**

  Finally, `run_read_omno2.sh` will find the last OMI_SP_yyyymmdd.mat file produced, the last albedo file retrieved, and run the MATLAB function necessary to import all satellite data between those two dates. This range was the simplest way to ensure that the necessary albedo file was retrieved.

5 Version Control

5.1 Repositories

The core code for BEHR is contained in a Git repository on our group's Synology DS1813+ NAS file server. If you are not familiar with Git, I recommend reading chapters 1-4 at `http://git-scm.com/doc`, which includes information on installing Git on your system as well as the basic commands. If you are working on a Windows machine you make also want to look at `http://guides.beanstalkapp.com/version-control/git-on-windows.html` and follow their recommendations, although as I've never used Git on a Windows machine, I can't say for certain how well that works. Once you have Git installed and working on your machine, navigate to the folder that will be your working directory for the project in either Terminal (Mac) or Command Prompt (Windows) and run the command:

```
git clone ssh://RCCohenLab128.32.208.13/volume1/share-sat/SAT/BEHR/BEHR_GitRepo.git
```

This will mirror the repository as a new folder in that directory. A second repository, at `128.32.208.13/volume1/share-sat/SAT/BEHR/MiscUtils.git` contains some general utility Matlab scripts that I have found useful. Some were downloaded from the Matlab file exchange, many were functions I found myself needing rather often. A third repository is `128.32.208.13/volume1/share-sat/SAT/BEHR/AircraftProfiles.git`, with functions to work with aircraft data sets. Additional repositories will be added to Table 1. I recommend that these be downloaded to folders called "BEHR", "Utils", and "NO2 Profiles" within your main Matlab directory (which you can check with `userpath` at the Matlab command prompt. This way, any internal links that I've set up to be robust as `fullfile(userpath,x,y,z,...,file)` should work smoothly on either a Windows or Mac platform.

Repo location	Rec. folder	Contains
`BEHR_GitRepo.git`	BEHR	All the code needed to run BEHR
`MiscUtils.git`	Utils	Miscellaneous utilities not needed for BEHR but generally helpful
`AircraftProfiles.git`	NO2 Profiles	Functions to work with aircraft data sets, including code to validate satellite data against such data sets.
`BEHR_MatlabClasses_GitRepo.git`	Classes	Certain MATLAB classes I wrote to help manage certain tasks (e.g. error messages)

Table 1: Summary of the IP addresses, recommended folders within the main Matlab directory, and contents of the three Git repositories.

This repository is in place now and will be kept relatively up-to-date as I progress. I will also try to remember to make a simple copy of the BEHR code to the file server which will not require the use of Git to obtain. However, there are several reasons you should consider learning to use Git if you haven't yet. (If you have, you probably stopped reading this as soon as I told you where the repository was.)

1. *It keeps everything in one place, and makes it easy to keep everything up to date.* As long as you make changes in the directory that the Git repo consists of (and commit the changes periodically), those changes are tracked. So, you can roll back to an earlier version if something breaks, or see what code you (or I) changed.

2. *Parallel development.* If multiple people are developing BEHR at one point, each person can create their own branch and develop simultaneously, while still being able to update the project on the server, and eventually merge the development lines together.

3. *Code sharing.* Conversely, if two (or more) people are both using BEHR, this makes it easier to synchronize code when and if you want.

5.2 Best practices for version control

Read this section before you start changing things! As I'm writing this, the version control for BEHR is fairly straightforward as I'm the only user on the project. If multiple users are developing the code simultaneously, adhering to some good practices will help keep everyone sane.

- **The master branch should only contain stable code.** Generally in Git, the "master" branch is just as it sounds: the main branch with reliable code. This branch should contain the version of BEHR that produces the *published website* data.
- **Changes to the master branch should be understood by all parties.** Before merging any work into the master branch, everyone involved in developing BEHR should sit down together and be sure they understand the changes to be merged. Ideally, this would include a full code review where everyone satisfies themselves that there are no lingering bugs; but realistically that probably won't happen.
- **Each student on the project should create their own development branch.** This will serve as a sort of "personal master branch," i.e. the branch that all the rest of your branches split from and that you merge back into. Ideally, only this branch should then be merged into the true master branch.
- **Not every local branch needs a remote branch.** Git is really useful because you can create these branches to develop new code while leaving a stable state alone. However, not every local branch needs to have a corresponding remote branch. To keep the remote repo from getting too crazy, only make remotes when you need to sync that branch across multiple computers.
- **Clean up remote branches when finished with them.** Once you merge or delete a development branch, make sure the remote that went along with it gets deleted.
- **Finally, don't check out someone else's dev branch without telling them.** And definitely don't push back to it, as then they have unexpected code coming into their dev branch.

Along with having good practices for version control, be sure to update the file `Changelog.txt` under the Documentation folder of the BEHR repository. This should contain itemized, human-readable descriptions of the change for each new version or revision. To help this, you should keep a "Development" section at the top of it where you can record significant changes as you make them. When this merges into the master branch, change the header from "Development" to the new version number. Make sure to post the updated change log on the website (it is uploaded as a regular text file and linked under the "Documentation" section of the "Download BEHR data" page on the website.

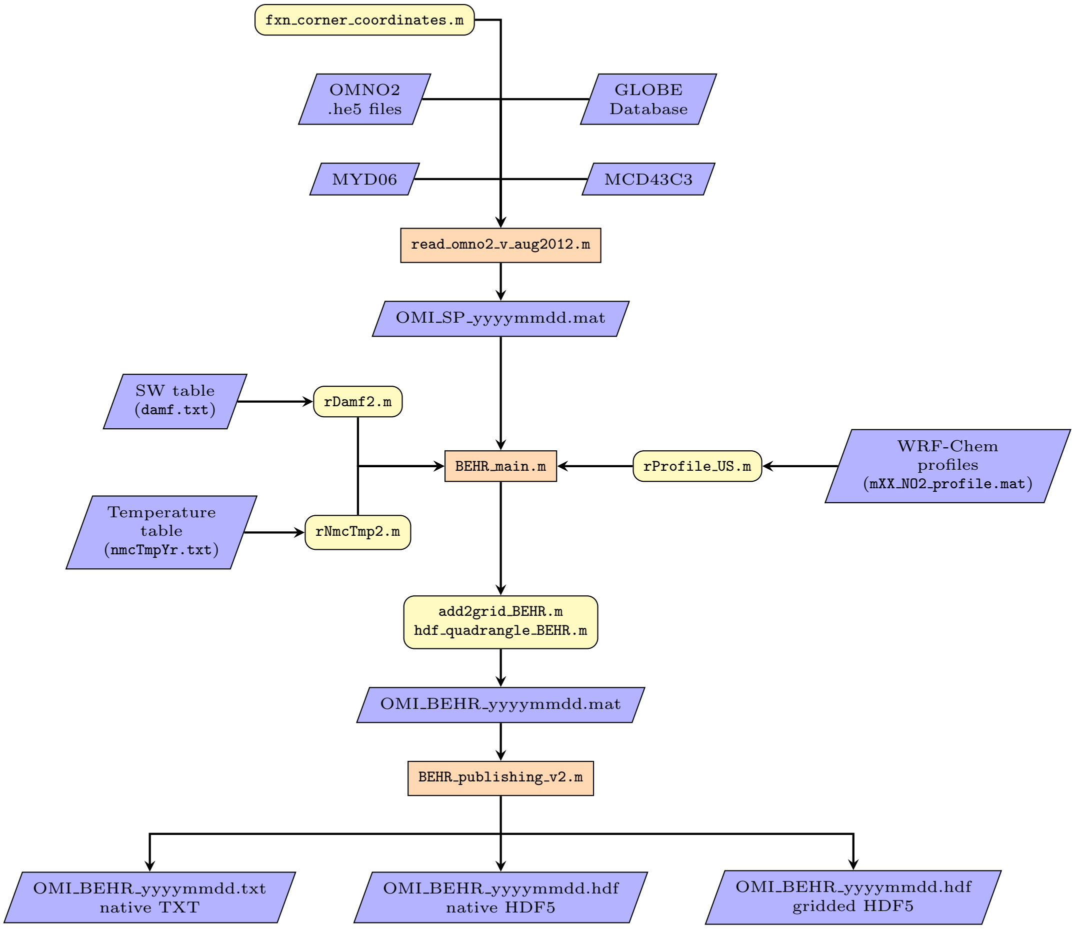

Figure 1: The structure of the BEHR program. Orange rectangles indicate the primary MATLAB functions that need to be executed; yellow rounded rectangles are other functions called internally by those main functions. Blue trapezoids represent input or output data.

6 Program structure

This section describes the key files of code in BEHR to make it run. Instructions for running it contained within this section are predicated on the idea that you're running BEHR on a desktop computer. For instructions on running it on a cluster, see §7.

The general program flow is outlined in Fig. 1. The first step is reading in the OMNO2 and ancillary data using `read_omno2_v_aug2012.m`. The result of this function is taken by `BEHR_main.m` which computes and applies the BEHR AMF, as well as grids the data to a 0.05°×0.05° grid. Finally the results can be published to the website using `BEHR_publishing_v2.m`.

6.1 Read data

The first step in running BEHR on new data is to read the NASA files into Matlab. This is done using the `read_omno2_v_aug2012.m` file. This is a Matlab function, but it is meant to be run from the editor without any input arguments. All of the properties that need to be set are coded into the function. However, it can accept start and end dates as inputs to the function, which can be useful to run it in batches.

The "`v_aug2012`" part of the name indicates that this file is intended to work with OMNO2 v. 3, which was released in Aug. 2012 (or at least the technical specs for it were).

This function serves several purposes:

1. It reads all relevant variables from OMNO2 files into Matlab.
2. It computes pixel corner points based on the field of regard of each pixel and the area over which 50% of the pixel's sensitivity can be attributed.
3. It averages the MYD06_L2 cloud product data to each OMI pixel.
4. It averages the MCD43C3 albedo product data to each OMI pixel.
5. It averages the GLOBE terrain data to each OMI pixel, converting from altitude to terrain pressure.

For each day that this function processes, a `.mat` file is saved with a single variable, `Data`. This variable is a data structure, in which each OMI swath for that day is stored under a different top-level index (i.e., `Data(1)` refers to the first swath of that day, and `Data(2)` the second, etc.). These data structures contain the data read from the OMNO2, MODIS, and GLOBE files as matrix fields. Each element of the matrix corresponds to an OMI pixel.

Production (i.e. not testing) files output from this script will generally have the name `OMI_SP_yyyymmdd.mat`. Filenames with additional information are generally testing files I have created in the course of various debugging runs, and should not be used to produce NO_2 data.

To run this file, enter the latitude and longitude boundaries for the area to retrieve (any OMI pixels outside this area will be discarded) and the date range to process. The first time you run this file, you may need to edit the paths to the various files; follow the instructions in the comments to identify which directory corresponds to which type of file. To specify a path to a folder on the lab server, Mac users should begin the path with `/Volumes`. PC users: the path should begin with the letter you chose when mounting the volume.

Should a new version of the OMNO2 files be released, lines 293 and 333–358 (dealing with various H5 loading functions to read OMNO2) might need updated to reflect any change in dataset names in the OMNO2 he5 files. The MYD06 and MCD43C3 files are loaded further down in the code; follow the comments.

6.2 Recalculate AMF and Tropospheric Column

This is handled by the `BEHR_main.m` function in `BEHR/BEHR_Main/`. Compared to the reading function, this is rather short, but it is the key component of BEHR. It goes through several steps:

1. It uses the TOMRAD look up table from NASA's OMNO2 product to generate box AMFs for each pixel, but using MODIS albedo and GLOBE terrain pressure. This is done for the clear and cloudy cases.
2. Reads in NO_2 profiles generated from WRF-Chem and bins them to each OMI pixel.

3. Calculates the full AMFs by combining the WRF profiles and box AMFs.

4. The new AMF is then applied to the tropospheric slant column, found by multiplying the NASA AMF with their vertical column.

5. Finally selected data fields are gridded onto a 0.05° × 0.05°.

The resulting outputs are saved in BEHR files that contain two variables: `Data` contains the original OMI pixel data, with the new BEHR AMF and NO_2 column density appended. `OMI` contains the variables gridded to 0.05° × 0.05°. Both retain the same structure wherein the top-level index represents a single swath. However, in `OMI`, every swath's grid covers the entire region of interest, and cells with no data for that swath have a fill value.

You may wonder that the grid is smaller than the regular OMI pixels; this technique is called *oversampling* and allows us to take advantage of the fact that the OMI pixels do not overlap exactly day-to-day to obtain an effective resolution greater than the pixel size if we average over longer time periods.

This function should require minimal upkeep even in the event a new NASA product is released.

6.3 Weight pixels and map

This step is used when creating maps of BEHR NO_2 data. The existing function to handle this is `no2_column_map_2014.m`. It can be called with a GUI by using `no2colmap_wrapper.m`. Either way, this uses the gridded BEHR data to do time averaging and `omi_pixel_reject.m` to filter out pixels with cloud contamination or that are affected by the row anomaly.

When doing temporal averaging, each grid cell has an areaweight associated with it; this is the sum of the reciprocal of the areas of each pixel that overlapped that grid cell. By weighting by the reciprocal of the area, it gives more weight to grid cells that got information from small, more representative, pixels. Therefore, any temporal average should be weighted by this field.

Plotting can be accomplished using the `m_map` package (which is used in the above functions, and which should be included in the BEHR Git repo). MATLAB also has built in mapping functions which can be used if desired. Generally a `pcolor` plot of some sort is the best option.

6.4 Publish

Once a production quality version of BEHR is ready to go, it needs to be published to the website (see §8). The data is published in three formats:

- Plain text CSV files that list the important variables for each pixel. The 2D structure of the swath is lost. This is considered at the native OMI pixel resolution.

- HDF5 files containing the same variables as the plain text files. However, because HDF files can store data in arrays with any number of dimensions, the 2D structure of the swath can be preserved. This is also at the native OMI pixel resolution.

- HDF5 files containing a subset of the variables at the 0.05° × 0.05° resolution.

The publishing is handled by the function BEHR_publishing_v2.m. It is set up to produce a single one of the above three types with each run. Which type and the dates to produce for can be specified as inputs to the function or by modifying the corresponding values in the "Set Options" part of the code and running it without any inputs. It also retains the ability to produce some specialized versions of the product, mainly those that used *a priori* profiles derived from the DISCOVER-AQ research flights.

7 Running BEHR on a compute cluster

7.1 BEHR in Matlab

7.1.1 Parallelization

Starting in Jan 2015, the main BEHR code (read_omno2_v_aug2012.m and BEHR_main.m) had simple parallelization added to the code body. This is only intended to be used when the code is run on a cluster because in order to run operations in parallel, Matlab must send all relevant variables from the main instance to the parallel "workers" actually executing the code. Given the size of the variables routinely used in BEHR, this communication overhead can result in overall slower work than a serial execution if too few cores are used. As of 29 Jan 2015, I have yet to run any sort of rigorous benchmarking tests, but anecdotally, running BEHR in parallel with only 2 cores seemed to result in a slower execution than running it in serial.

There were numerous small changes to the code to enable parallelization, most importantly, the for loop over days was replaced with a parfor loop. parfor in Matlab allows multiple iterations of a for loop to run in parallel, and automatically handles the distribution of data. Compared to an spmd block, we give up control over how and when the data is distributed in exchange for Matlab handling it automatically.

Compared to previous versions of BEHR, most of the changes in the code (too numerous to list individually) were mainly to allow Matlab to "slice" variables appropriately for inclusion in a parfor loop. The change that is significant for the user is that a number of global variables were added to control both the action of the parallel loop and the paths to data.

The reason these variables were made global variables instead of inputs to the function was to allow them to be set in a run script once. This makes the run script more "bash like" (programs compiled from a shell like bash—including GEOS-Chem and WRF-Chem—often reference environmental variables to determine how they compile).

The two variables controlling the action of the parallel loop are onCluster and numThreads. onCluster should be set to true in a run script whenever the script is being executed on a cluster. numThreads is, by default, set up in the run script template to take its value from an environmental variable, MATLAB_NUM_THREADS set in the bash shell that calls the Matlab instance running it. This was done because it is possible to set this environmental variable by referencing another that relates directly to how many CPU cores are available. See the example in §7.1.3 for an example.

The path variables are all defined in the .m files, and are set up such that if `onCluster` is true, the functions will look to global variables setup in the run script for those paths. (If any aren't defined, an error is thrown.) This is so that you only have to edit the run script to use BEHR on a cluster. If `onCluster` is false, the function looks to the paths coded into the functions.

7.1.2 Running it

The code was set up to be executing using a "runscript," which is a Matlab script file that can be called from the command line as:

```
matlab -nosplash -nodisplay -r "run('runscript.m')"
```

The argument `-nosplash` tells Matlab not to show the splash screen on startup, and `-nodisplay` tells Matlab that it shouldn't start the GUI interface. (Since we're working from the command line, we don't need it, and we don't want it to try to open it if we can't see it anyway.) `-r "..."` tells Matlab to execute the command given in quotation marks as soon as it starts up. In this case, that is to execute the runscript in the current directory.

A template for a BEHR runscript can be found in `BEHR/Run_Scripts/`. You'll notice that the runscript template does several things:

1. Sets the global variable `onCluster` to `true`. This lets any scripts that use that variable know that it is running on a cluster and should activate any parallel elements in the code.

2. Sets the variable numThreads. By default this is set to the variable of the environmental variable `$MATLAB_NUM_THREADS` from the shell that executed this instance of Matlab. More on why this is preferable below.

3. Detects any active parallel pools and shuts them down before exiting.

4. Everything is wrapped in a `try-catch` block that will, in the event of an error, prints the error information to the console, closes any active parallel pools, and exits with status code > 0.

This is a very general template, so you can use it to call any function you parallelize using the global variables `onCluster` and `numThreads`. For BEHR, you'll also have to set all the global path variables for `read_omno2_v_aug2012.m` and `BEHR_main` in the runscript.

7.1.3 Submitting to the cluster queue

Like any job you want to run on a computing cluster, you'll need to write a shell script that is put into the queue for the cluster. Since the Savio cluster that I use operates with the SLURM scheduler, this section will be written from the point of view of submitting to SLURM using the bash shell. (I assume that if you use tcsh or another shell that you know what you're doing well enough to make the necessary adjustments.) Below will be an example submit script, each important line will be described afterward.

```
1  #!/bin/bash
2  #
3  # Job Name:
4  #SBATCH --job-name=BEHR
5  #
6  # Partition:
7  #SBATCH --partition=savio
8  #
9  # Account:
10 #SBATCH --account=ac_aiolos
11 #
12 # QoS:
13 #SBATCH --qos=condo_aiolos
14 #
15 # Number of nodes and processors per node
16 #SBATCH --nodes=1
17 #SBATCH --ntasks-per-node=20
18 #
19 # Wall clock limit:
20 #SBATCH --time=24:00:00
21 #
22 ## Run command
23 export MATLAB_NUM_THREADS=$((SLURM_NTASKS_PER_NODE-1))
24 cd /global/home/users/<me>/<behr_run_dir>
25 module load matlab
26 matlab -nosplash -nodisplay -logfile "runlog.txt" -r "run(’runscript_behr.m’)"
27 MATLAB_EXIT=$?
28 exit $MATLAB_EXIT
```

Line 1 Lines starting with a `#!` are called a *shebang* in bash-speak, this one tells the computer to run the script using the bash shell. Including this is a safety measure; it ensures the script is always run using bash if another shell interpreter is active.

Line 2 A `#` indicates a comment in bash

Line 4 The `#SBATCH` at the beginning of this line tells the SLURM scheduler that a setting is being passed. Bash ignores it because of the `#`, but SLURM does not. In this case, we're setting the name of the job that will appear in the queue.

Line 7 Tells SLURM which partition of nodes to run on. We use “savio” unless we need lots of RAM.

Line 10 The SLURM account that would be charged for running (I think). Ron is under the “aiolos” account; the “ac” indicates that it is the account.

Line 13 “QoS” determines what rules the job is run under and how compute time is charged. The part after the underscore will always be the account, “aiolos.” The part before the underscore determines the rules. “condo” means that we can use up to 8 nodes at a time and won't be charged for it.

Line 16 How many nodes to to. A node is the computing unit of the cluster—each node will only run one job at a time.

Line 17 How many cores to use on each node. Each node has two 10-core processors for 20 cores maximum per node.

Line 20 How long to let the job run before forcing it to quit. Here, we set it to 24 hours. It's good practice to do this to prevent a job from running indefinitely.

Line 23 `export` in bash means "set this as an environmental variable" which programs executed from this shell can access. The `$(( ))` in bash means to evaluate an arithmetic expression and return the result. So here we're saving one less than the number of tasks per node in the `MATLAB_NUM_THREADS` variable. I do this to be a little bit careful to leave a core free for the main Matlab instance, although I don't know if that's strictly necessary.

Line 24 Change to our run directory (just in case the job starts somewhere else). `<me>` and `<behr_run_dir>` are just placeholders.

Line 25 Savio organizes applications by modules, here load the matlab module to be able to run matlab.

Line 26 Execute matlab with command line arguments. The only new one is `-logfile "runlog.txt"` which saves all Matlab command window output to the file runlog.txt. One mistake to avoid is including the `-nojvm` flag, because (apparently) the parallel computing toolbox needs Java to work. Don't ask me why.

Line 27 The `$?` is the last given exit code. We save this to a variable...

Line 28 ...and then exit this script with that exit code. This will let you know if the script succeeded or failed.

If this script is named e.g. `matrun` then typing `sbatch matrun` on the cluster will submit it to run. You can check the status of all jobs with `squeue`.

The reason that we use the variable `MATLAB_NUM_THREADS` to pass the number of available cores to the run script is to be a little careful about not causing Matlab to request more cores than we've set aside. By deriving `MATLAB_NUM_THREADS` from the SLURM variable indicating the number of cores to be used per node, we ensure that a change to the SLURM settings is propagated through to our Matlab instance without any intervention on our part.

7.2 Resources

1. Matlab documentation on parfor loops: `http://www.mathworks.com/help/distcomp/parallel-for-loops-parfor.html`

2. Matlab parfor loops, classification of variables: `http://www.mathworks.com/help/distcomp/classification-of-variables-in-parfor-loops.html`

3. The High Performance Computer (HPC) User Guide: `http://research-it.berkeley.edu/services/high-performance-computing/user-guide`

4. SLURM Documentation: `https://computing.llnl.gov/linux/slurm/documentation.html`

5. List of SLURM parameters (i.e., what can be set in the bash run script on the lines beginning with `#SBATCH`: `https://computing.llnl.gov/linux/slurm/sbatch.html`

8 Maintaining the website

8.1 Gaining administrative access

To be able to edit the website, you'll need to make an account. At the top right of every page, there should be a register link. Complete that to create an account.

Once you have an account, you'll need to contact an existing administrator to have your account promoted. Existing administrators are (try in order):

- Josh Laughner (joshlaugh5@gmail.com)
- Anna Mebust (annamebust@gmail.com)
- Ashley Russell (ashley.ray.russell@gmail.com)

Go ahead and add yourself to the top of this list once you've been promoted. If none of the existing admins can help, you can also contact Stephen dos Remedios (steven@meetthere.com), he is the one who helped set up the website initially, and can help with most issues, although he usually prefers to have one of the other admins handle adding new admins.

8.2 Editing the content

Log in with your newly promoted admin account. At the top of the site, you should now see the Dot Net Nuke Community bar. To edit, change the dropdown menu at the top right from "View" to "Edit." Each section in each page should now have a "Manage" ghost button when you hover over it. Hovering over that in turn brings up a menu, one option is "Edit content." This is how you modify the text and most of the content in the website.

Some small files should be uploaded directly to the website, things like the change log, user guide, etc. In the edit content window, the planet with a chain link button is the hyperlink manager. This lets you insert links to other websites, but also to uploaded files. The button next to the URL field bring you to the Document Manager, where such files can be uploaded. *This is not how to upload new versions of BEHR, see §8.3!*

8.3 Providing new data

The website is set up such that a URL pointing to `/behr` (no `http://` or `www`, this is a local path) points to the directory `/volume1/share-sat/SAT/BEHR/WEBSITE/webData/` on the file server at 128.32.208.13. The most important subdirectories are noted in Table 3.

To update the publicly available produce, simply replace the files in these directories with the proper output files from `BEHR_publishing_v2.m`. No other action is necessary.

Website URL	Server folder
/behr/behr_hdf	/volume1/share-sat/SAT/BEHR/WEBSITE/webData/behr_hdf
/behr/behr_txt	/volume1/share-sat/SAT/BEHR/WEBSITE/webData/behr_txt
/behr/behr_regridded_hdf	/volume1/share-sat/SAT/BEHR/WEBSITE/webData/behr_regridded_hdf

Table 3: Important data paths on the website and file server.

9 Additional utilities

9.1 Verification

10 Current development

The current focus of work on BEHR is to give it daily (rather than monthly average) *a priori* NO_2 profiles. To do so, I've completely rewritten how the NO_2 profiles are imported into `BEHR_main.m`.

The new pieces of code are `rProfile_WRF.m`, which replaces `rProfile_US.m` and reads profiles from the `WRF_BEHR_*_yyyy-mm-dd.nc` netCDF files instead of the .mat files used previously. These netCDF files are produced by the Bash shell script `slurmrun_wrf_output.sh` or `run_wrf_output.sh`. Ideally the difference between these two is that `run_wrf_output.sh` is meant to run in serial, while `slurmrun_wrf_output.sh` runs in parallel, using the `srun` command with the SLURM scheduler. However, development of `slurmrun_wrf_output.sh` tends to proceed more rapidly, so it is preferred although it can only be run on the Savio cluster or another cluster using the SLURM scheduler.

Both scripts use the NCO tools. These are installed on the Savio cluster (although as a module that needs to be loaded) and can be installed on your computer if desired. Essentially, `slurmrun_wrf_output.sh` determines how to average the WRF output (monthly, daily, hourly), which set of variables to put in the final files, and whether it is preparing *a priori* for OMI or TEMPO (the TEMPO one is a notional implementation at current, making certain assumptions about the operation of TEMPO). It identifies the files necessary and instantiates multiple instances of the appropriate `read_wrf_output.sh` (may also be `read_wrf_tempo.sh` or `read_wrf_emis.sh`) to allow parallel execution. These in turn make use of built-in NCO functions and several NCO scripts I've written to both extract the key variables and calculate several additional quantities not automatically included in the WRF output. Cutting down the files in this manner substantially reduces the amount of data that needs to be moved from a cluster capable of running WRF-Chem to whatever computer is running BEHR.

This information is extracted from the raw WRF-Chem output files, `wrfout_d0X_yyyy-mm-dd_hh:mm:ss`. My scripts assume that there is one output "frame" per file, that is, each file represents a single slice in time. It is possible that WRF-Chem might save multiple time slices to each file, although the resulting files will likely be huge.

`rProfile_WRF.m` is also fairly intelligent about how it reads and applies the NO_2 profiles,

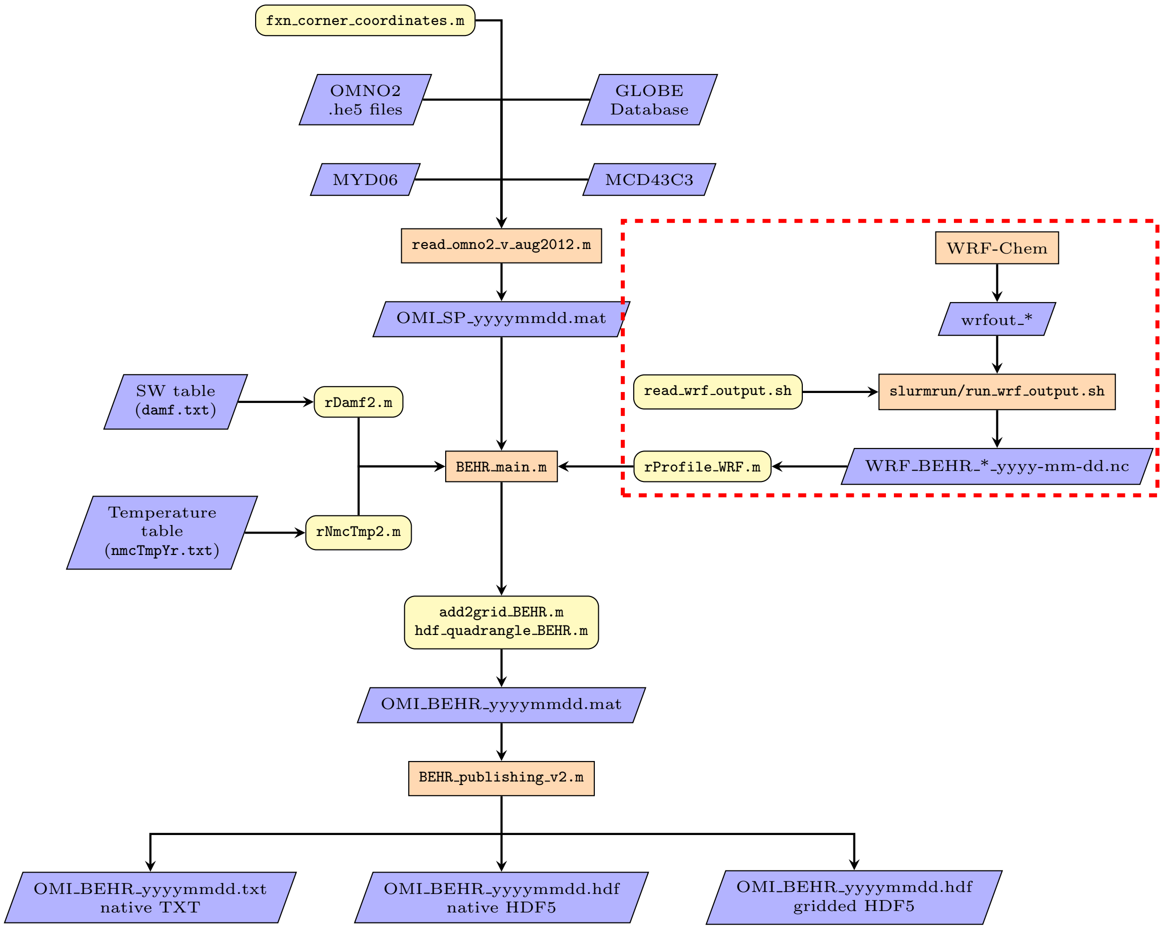

Figure 2: The changes necessary to allow the use of daily *a priori* profiles, marked by the red box.

accounting for different spatial resolutions as well as being able to use different temporal resolutions. However to do the latter, it expects the profiles to be organized in subfolders named by the temporal resolution of the *a priori* profiles in them: “hourly,” “daily,” or “monthly.”

Appendices

A Running downloads in the background

B Setup for automatic downloads

B.1 OMNO2

Sources:

- `http://disc.sci.gsfc.nasa.gov/additional/faq/general_user_services_faq.html#subscription`

- `http://disc.sci.gsfc.nasa.gov/additional/scienceTeam/s4pa_mri.html`

Automatic downloads for OMNO2 are the trickiest part of this because they require a data subscription in order to be downloaded automatically. I chose to use an inactive subscription in which our computer makes a request, receives the list of URLs, and executes the download. This gives us the maximum control over when the download process occurs.

Because of the nature of these systems, I recommend that automatic downloading be carried out on an Ubuntu or other Linux-based OS. These tools are easiest to configure in such an OS, and these instructions are meant for those.

1. The computer must have **openssh** installed. I will be very surprised if it doesn't, but you can make sure by checking that there is a folder `/usr/lib/openssh`. If not, install with:

   ```
   sudo apt-get install openssh
   ```

2. The way that the machine request interface works is that upon a request (delivered by wget'ing a formatted URL), it will produce a delivery notice with the URLs to the files requested via secure file transfer protocol (sftp) to the designated computer at a fixed IP address. I have the users `nasa` set up on the black Compaq computer at 128.32.208.13 for this purpose.

3. For security, the computer has password authentication for SSH disable, only asymmetric key login allowed. Authorized public keys need to be pasted into the `~/.ssh/authorized_keys` file (one per line) for the correct user. The OMNO2 DISC sent their public key to me upon request, this should go in the `authorized_keys` file for the `nasa` user. (Password authentication is disabled by changing the line in `/etc/ssh/sshd_config`

   ```
   PasswordAuthentication yes
   ```

   from `yes` to `no`. If there is a `#` sign at the front of the line, remove it.)

4. To set up sftp, find the line in `/etc/ssh/sshd_config` containing:

```
Subsystem sftp
```

Change it and add lines following it to match:

```
Subsystem sftp /usr/lib/openssh/sftp-server
Match group ftpaccess
ChrootDirectory %h
X11Forwarding no
AllowTcpForwarding no
ForceCommand internal-sftp -d /www
```

This will require the user to be in the group `ftpaccess` (next step), lock it to its home directory, and automatically start logged into the `www` directory in its home folder.

5. Add the `nasa` user to the ftpaccess group

   ```
   sudo groupadd ftpaccess
   sudo usermod -a -G ftpaccess nasa
   ```

   Also create a `www` directory in the `nasa` user's home folder if it does not already exist.

This should be sufficient. You may test this by adding your own public key to the `nasa` user's authorized keys file and attempting to sftp in:

```
sftp nasa@128.32.208.11
```

If successful, you should be in a usual ftp prompt in the www directory.

B.2 MODIS

Sources:

- `http://modaps.nascom.nasa.gov/services/faq/`

MODIS requires no special setup to establish a subscription, as it has an open ftp server. We access it using a combination of `wget` and SOAPpy.

B.3 General setup

All the necessary scripts are written in Bash or Python and are contained in the BEHR repo's Downloading folder. These can be automated through the use of crontab. Most can be run from your user, however some need to be run as root (to allow access to the nasa user).

First, the root crontab, edit with `sudo crontab -e`:

```
# m h  dom mon dow   command
30 3 * * 6 /home/josh/Documents/MATLAB/BEHR/Downloading/order_omno2.sh
```

And the regular user's crontab, edit with `crontab -e`:

```
# m h  dom mon dow   command
0 1 * * 6 /home/josh/SatDL/get_modis.sh
0 1 * * 7 /home/josh/SatDL/get_omno2.sh
0 1 * * 1 /home/josh/SatDL/run_read_omno2.sh
0 1 * * 2 /home/josh/SatDL/run_behr_main.sh
0 1 * * 3 /home/josh/SatDL/run_publishing.sh
```

This reads as, e.g. "run order_omno2.sh at 3:30 AM on Saturday morning." The only two key points for the order are:

1. order_omno2.sh must run before get_omno2.sh.
2. get_omno2.sh and get_modis should run before any of the run_xxx.sh files.

These scripts rely on the following environmental variables and aliases being set. Add the following to your ~/.bashrc file:

```
# Define the environmental variables and aliases first,
# this way they'll be defined before the check if the shell
# is interactive

# User defined environmental variables
export OMNO2DIR="/mnt/sat/SAT/OMI/OMNO2/download_staging"
export MODDIR="/mnt/sat/SAT/MODIS"
export SPDIR="/mnt/sat/SAT/BEHR/SP_Files_2014"
export BEHRDIR="/mnt/sat/SAT/BEHR/BEHR_Files_2014"
export AMFTOOLSDIR="${HOME}/Documents/MATLAB/BEHR/AMF_tools"
export NO2PROFDIR="/mnt/sat/SAT/BEHR/Monthly_NO2_Profiles"
export HDFSAVEDIR="/mnt/sat/SAT/BEHR/WEBSITE/webData"
export MATRUNDIR="$HOME/Documents/MATLAB/Run"

# Aliases
alias startmatlab="export MATLAB_DISPLAY=0; /usr/local/MATLAB/R2014b/bin/

# Add to PATH
export PATH="/usr/local/bin:$PATH"
```

To the root .bashrc in /root/.bashrc (you'll need to use sudo to edit it) add the OMNO2DIR variable:

```
# User defined environmental variables
export OMNO2DIR="/mnt/sat/SAT/OMI/OMNO2/download_staging"
```

just before the following lines:

```
# If not running interactively, don't do anything
[ -z "$PS1" ] && return
```

Save this but DO NOT exit before testing that you can still make yourself root with sudo su.

To allow automatic emailing of success or failure, link or copy the `automessage.sh` script to `/usr/local/bin` (making sure it's executable, `chmod u+x automessage.sh`) and install the sendEmail package (`sudo apt-get install sendEmail`) if it is not already. Finally, place the following information in the file `.gmailcredentials` in both your home folder and `/root`:

```
behrautodownload@gmail.com
!%e^XN!8Lk-j=HYRM-2S&wfvky&T5U+&
```