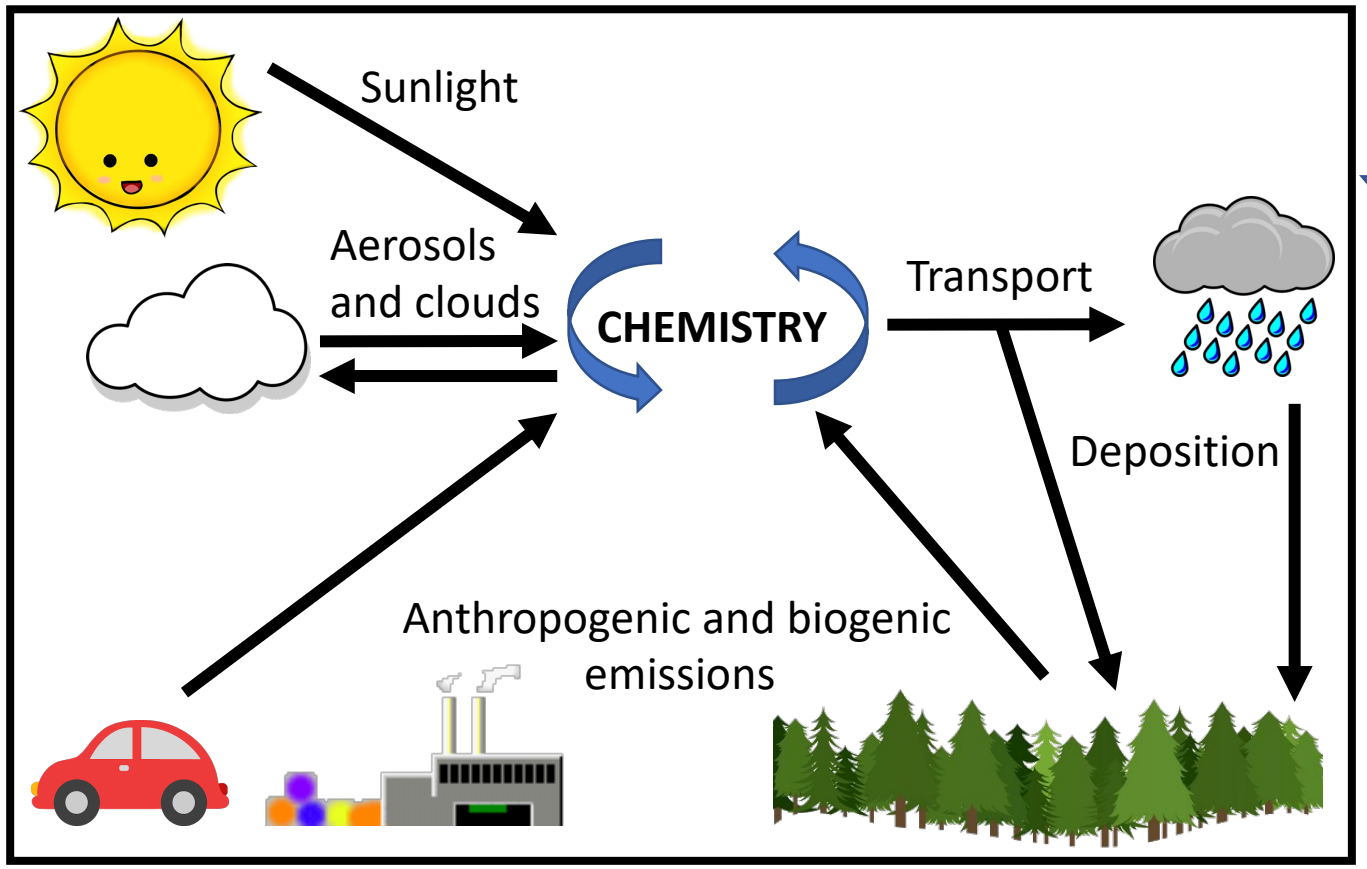
A generalized framework for estimating pollutant emissions and concentrations

Developing the CHEEREIO tool for chemical data assimilation

9 June 2022
IGC10
Drew Pendergrass and Daniel Jacob

Background: minimum ensemble values for weekly 2x2.5 global TROPOMI methane assimilation

Data assimilation allows atmospheric models to incorporate observations



Remote sensing

Assimilate

Emissions and chemistry corrected to match reality

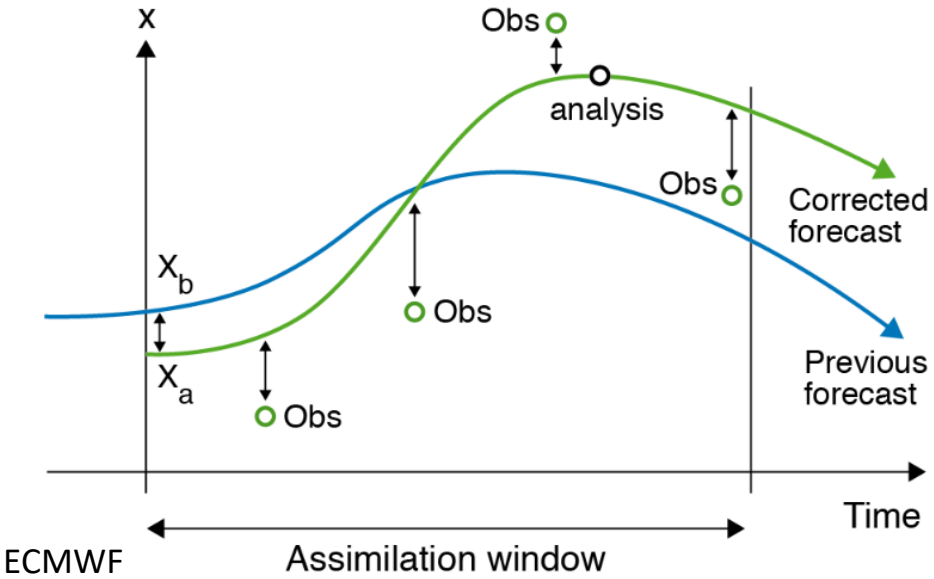
CHEEREIO

Predict

Surface monitoring

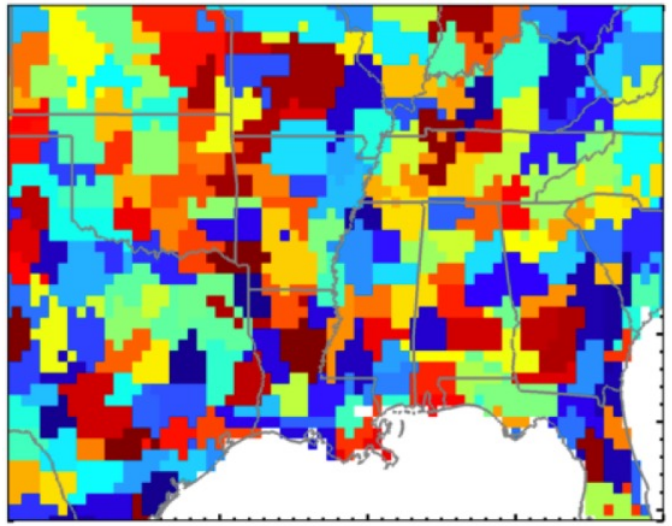
There are many approaches to assimilation and emissions correction

Variational approaches iteratively approximate optimal solution…



…but 4D-Var requires adjoint; harder to work with most recent GC chemistry

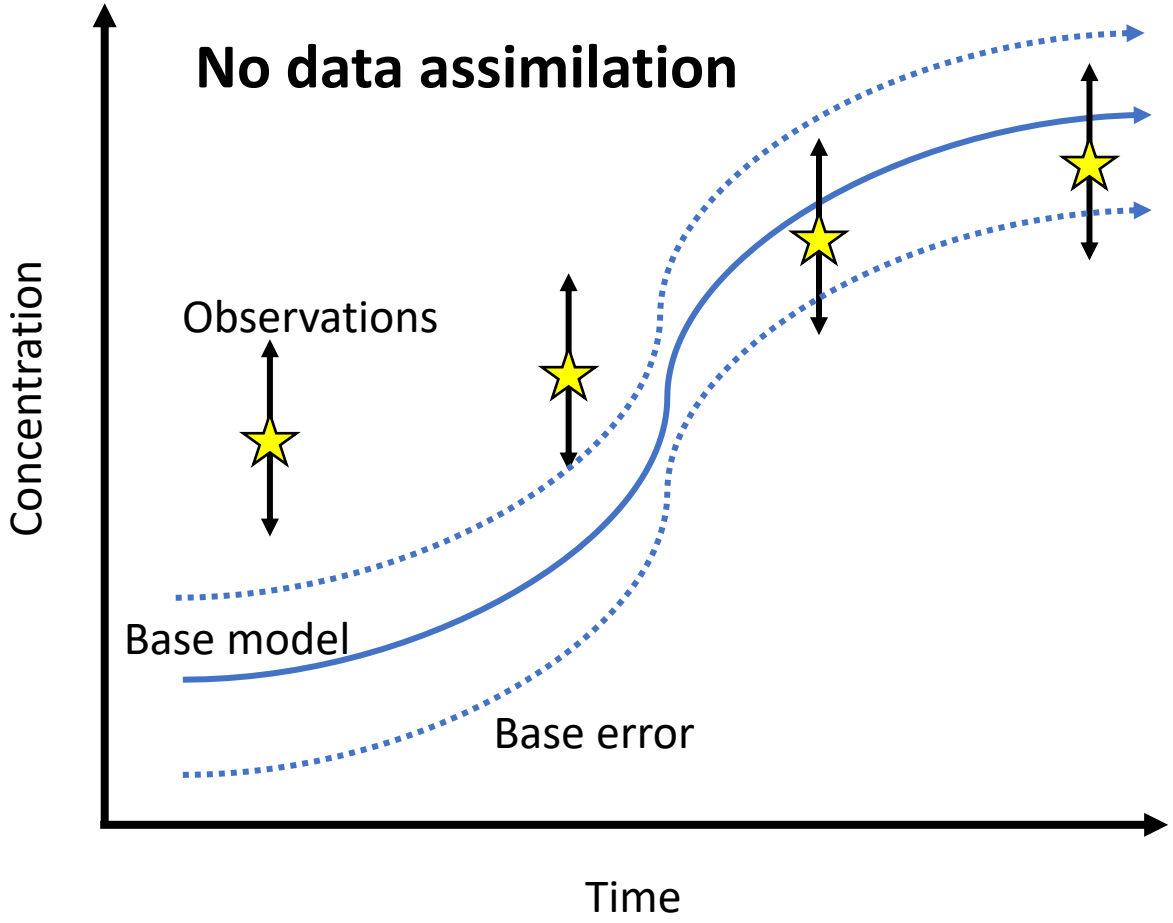
The **analytical inversion** perturbs clusters of grid cells to obtain full error characterization…

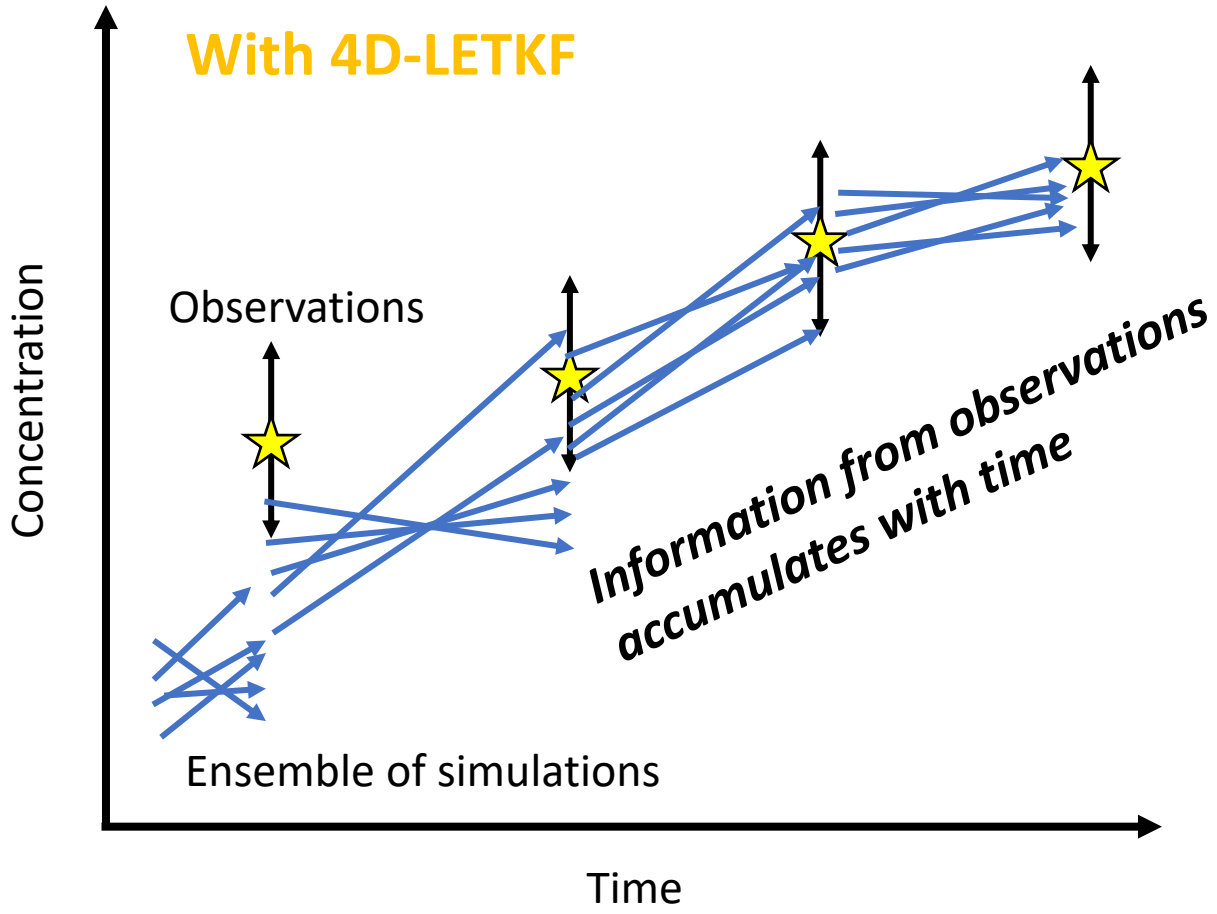


Sheng et. al., 2018

…but requires hundreds of runs, and works only for linear problems

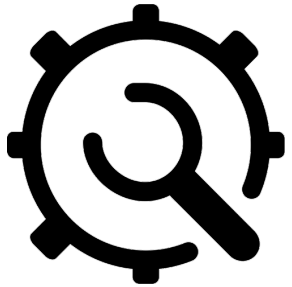
CHEEREIO uses ensembles with random emissions to emulate model uncertainty





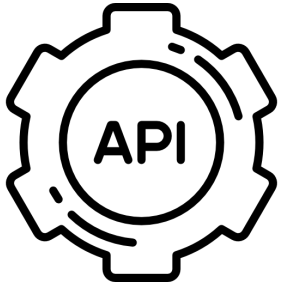
Builds on work already done by Kazuyuki Miyazaki (JPL) and others in the community

Summary of CHEEREIO project goals



Customization: Assimilate anything, in any GEOS-Chem configuration or simulation.

Maintainability: Science automatically aligned with latest model version



Powerful API: Minimal programming required for new experiments

Easy deployment: One configuration file controls installation and settings

The CHEEREIO workflow

Emissions, boundary conditions, and meteorology on disk

ens_config.json

Observations on disk

Sample prior distribution of scaling factors

Create template run directory (User modifies as necessary)

Spinup run(s)

Initial conditions

N ensemble members

Run GC until assimilation time

Wrap GEOS-Chem output, scaling factors, and observations in Python objects

GEOS-Chem/CHEEREIO model pipeline

No modifications of model code!!

Gather posterior concentrations and emissions

Simulate modeled observations (e.g. with averaging kernel)

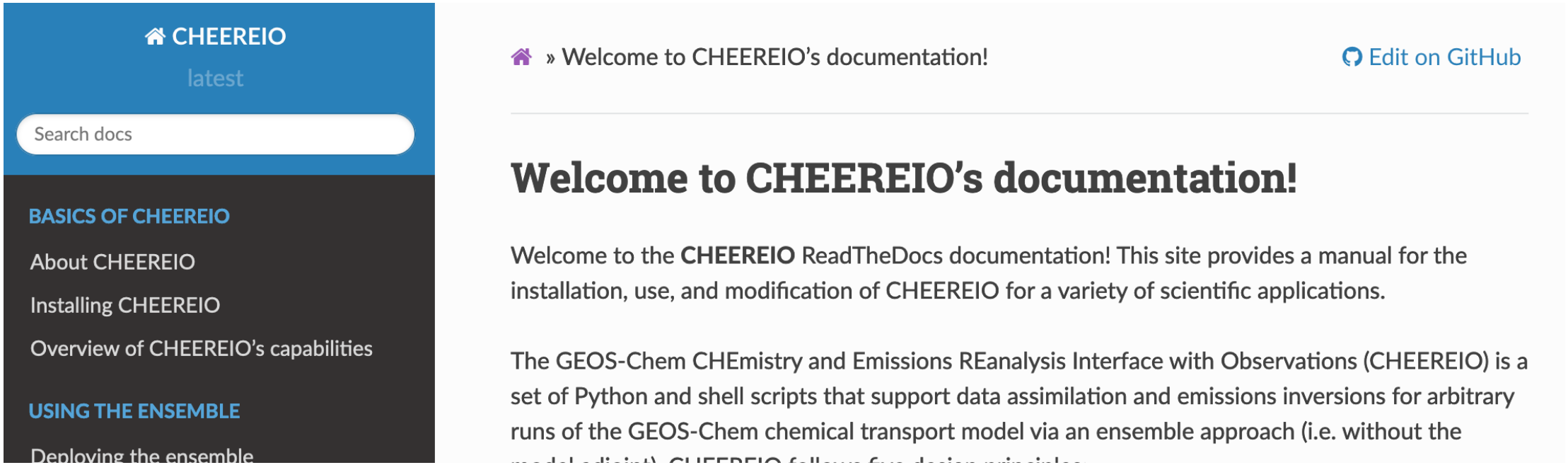
Produce user-defined **state vector** containing relevant emissions and concentrations

Subset LETKF output using user-defined **control vector**

Run assimilation algorithm (LETKF)

CHEEREIO assimilation pipeline

CHEEREIO is documented on ReadTheDocs for ease-of-use, with lots of examples!



https://cheereio.readthedocs.io

Total flexibility in the configuration file

```
"STATE_VECTOR_CONC" : [
    "NO",
    "NO2",
    "HNO3",
    "HNO4",
    "PAN",
    "MPAN",
    "N2O5"
],
"CONTROL_VECTOR_CONC" : [
    "NO",
    "NO2",
    "HNO3",
    "HNO4",
    "PAN",
    "MPAN",
    "N2O5"
],
"CONTROL_VECTOR_EMIS" : {
    "NOx":["NO","NO2"]
},
"HistorySpecConcToSave" : [
    "NO",
    "NO2",
    "HNO3",
    "HNO4",
    "PAN",
    "MPAN",
    "N2O5"
],
```

NO_X experiment

Quickly edit state and control vectors

```
"STATE_VECTOR_CONC" : [
    "CH4"
],
"CONTROL_VECTOR_CONC" : [
    "CH4"
],
"CONTROL_VECTOR_EMIS" : {
    "CH4":"CH4"
},
"HistorySpecConcToSave" : [
    "CH4"
],
```

CH_4 experiment

Observations easy to customize

```
"OBSERVED_SPECIES" : {
    "NO2_TROPOMI": "NO2"
},
"OBS_4D" : [
    "True"
],
"OBS_TYPE_TROPOMI" : [
    "True"
],
"TROPOMI_dirs" : {
    "NO2" : "/n/holylfs05/LABS/jacob_lab/dpendergrass/tropomi/NO2/2019"
},
```

NO_X experiment

```
"OBSERVED_SPECIES" : {
    "CH4_TROPOMI": "CH4"
},
"OBS_4D" : [
    "True"
],
"OBS_TYPE_TROPOMI" : [
    "True"
],
"TROPOMI_dirs" : {
    "CH4" : "/n/holylfs05/LABS/jacob_lab/dpendergrass/tropomi/CH4"
},
```

```
"TROPOMI_CH4_FILTERS" : "True",
"TROPOMI_CH4_filter_blended_albedo" : "0.75",
"TROPOMI_CH4_filter_swir_albedo_low" : "0.05",
"TROPOMI_CH4_filter_swir_albedo_high" : "0.4",
"TROPOMI_CH4_filter_winter_lat" : "50",
"TROPOMI_CH4_filter_roughness" : "60",
"TROPOMI_CH4_filter_swir_aot" : "0.1",
```

CH_4 experiment

Add new observation operators easily with Python-based inheritance

If you inherit from this abstract class…

```
class Observation_Translator(object):
    def __init__(self,verbose=1):
        self.verbose = verbose
        self.spc_config = tx.getSpeciesConfig()
        self.scratch = f"{self.spc_config['MY_PATH']}/{self.spc_config['RUN_NA
    #The globObs function returns a dictionary of observations that are within
    #that take place at a user-specified interval. The inherited function must
    #The returned dictionary must have keys for "latitude", "longitude", and "
    #where UTC time is an ISO 8601 date time string
    def getObservations(self,species,timeperiod, interval=None, c
        raise NotImplementedError
    #The function that gets the comparison between GEOS-Chem and
    #must have this signature and return a list in a highly speci
    def gcCompare(self,species,OBSDATA,GC,GC_area=None,saveAlbedo
        raise NotImplementedError
```

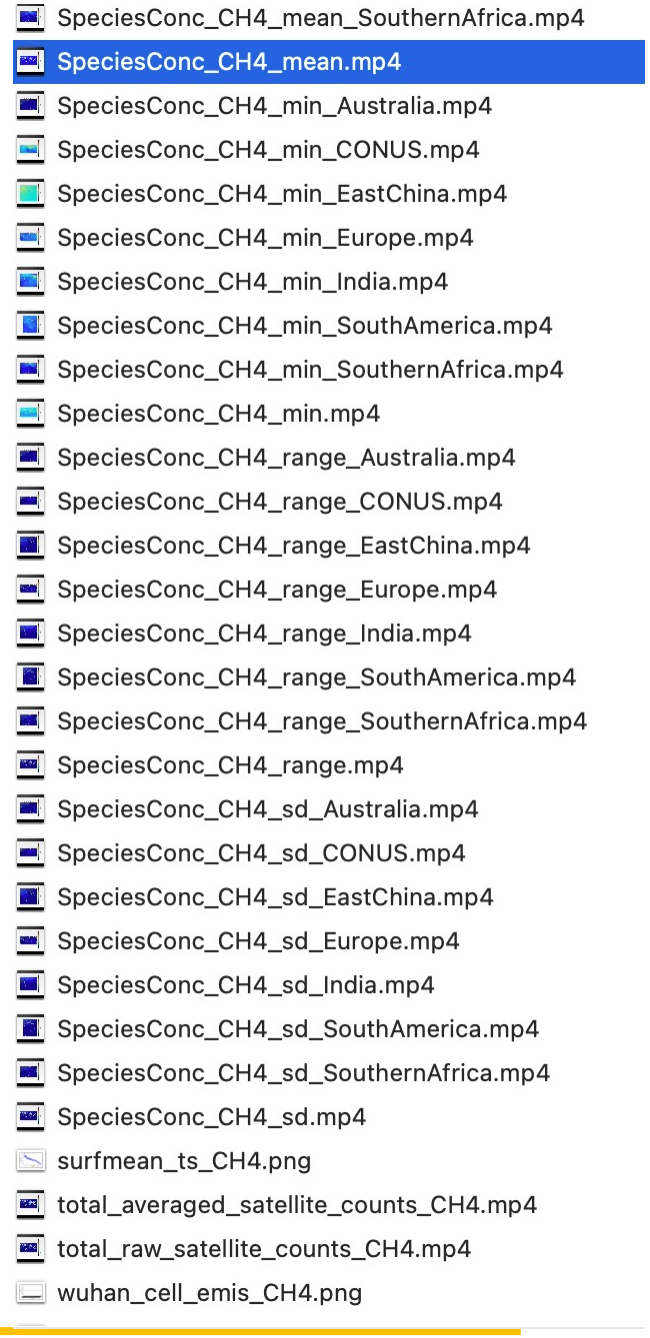
…then your operator works automatically!

```
class TROPOMI_Translator(obsop.Observation_Translator):
    def __init__(self,verbose=1):
        super().__init__(verbose)
    #Save dictionary of dates for later use
    def initialReadDate(self):
        sourcedirs = self.spc_config['TROPOMI_dirs']
        TROPOMI_date_dict = {}
        for key in list(sourcedirs.keys()):
            sourcedir = sourcedirs[key]
            obs_list = glob(f'{sourcedir}/**/S5P_*.nc', recursive=True)
            obs_list.sort()
            TROPOMI_date_dict[key] = {}
```

Postprocessing tools consolidate data and automatically produce hundreds of customizable animations and plots

No user code required: CHEEREIO infers the plots you want from the configuration file, but easy to modify and use postprocessing API for further customization

```
def makeDatasetForEnsemble(ensemble_dir,species_names,timeperiod=None,hourlysub = 6,subset_rule = 'SURFACE',
    subdirs,dirnames,subdir_numbers = globDirs(ensemble_dir,includeOutputDir=True)
    array_list = []
    for subdir in subdirs:
        print(f'Processing {subdir}')
        array_list.append(makeDatasetForDirectory(subdir,species_names,timeperiod,hourlysub,subset_rule))
    ds = xr.concat(array_list,'Ensemble')
    ds.assign_coords({'Ensemble':np.array(subdir_numbers)})
    if fullpath_output_name:
        ds.to_netcdf(fullpath_output_name)
    return ds
```



Automated testing implemented with Pytest: find bugs and broken code with one command!

```python
#These tests ensure that we are subsetting columns correctly in the GC_Translator class.

#From the methane restart, localize the state vector about 10,10 then get the column from the localization.
#Compare this to the true column from the file
def test_col_subset_of_localized_state_vector_methane():
    #override ens_config so that we are set to interpret the data properly
    testing_tools.setupPytestSettings('methane')
    #Get the column from the localized state vector
    gt = GC_Translator('data_for_tests/METHANE_TEST/TEST_0001/','20190101_0000',computeStateVec = True)
    locstatevec = gt.getStateVector(10,10)
    columninds = gt.getColumnIndicesFromLocalizedStateVector(10,10)
    column_from_statevec = locstatevec[columninds] #This column will have a scaling factor for the last entry. Remove it.
    column_from_statevec = column_from_statevec[0:-1] #Cuts off last entry
    ds = xr.load_dataset('data_for_tests/METHANE_TEST/TEST_0001/GEOSChem.Restart.20190101_0000z.nc4')
    da = np.array(ds[f'SpeciesRst_CH4']).squeeze()
    column_from_file = da[:,10,10]
    assert np.allclose(column_from_statevec,column_from_file,atol=1e-10)
```

Runs out-of-the-box, with included test files

Operates in a custom environment for reproducible results

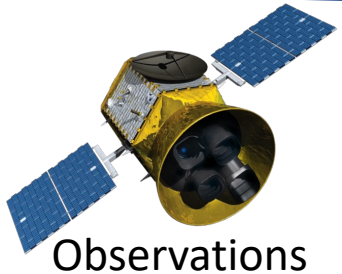
Know in seconds how badly you ruined the code!

```
(cheerio) [huce-r940 tests]$ pytest
============================================== test session starts ==============================================
platform linux -- Python 3.8.5, pytest-7.1.2, pluggy-1.0.0
rootdir: /n/home12/drewpendergrass/CHEEREIO/tests
collected 2 items

test_GC_Translator.py ..                                                                              [100%]

=============================================== 2 passed in 0.95s ===============================================
```

Applications currently under development

Optimize NO_x emissions in Korea

Data on disk



GEOS Chem

HEMCO

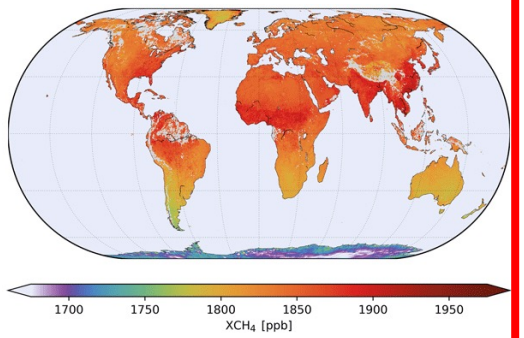
Observations → TROPOMI NO_2 columns and surface data

Model state → Species in NO_x mechanism

Emissions → NO_x scaling factors

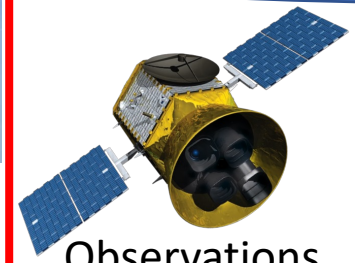
CHEEREIO

Simultaneously optimized NO_x scaling factors and concentrations



Optimize global CH_4 emissions

Data on disk



GEOS Chem

HEMCO

Observations → TROPOMI CH_4 columns

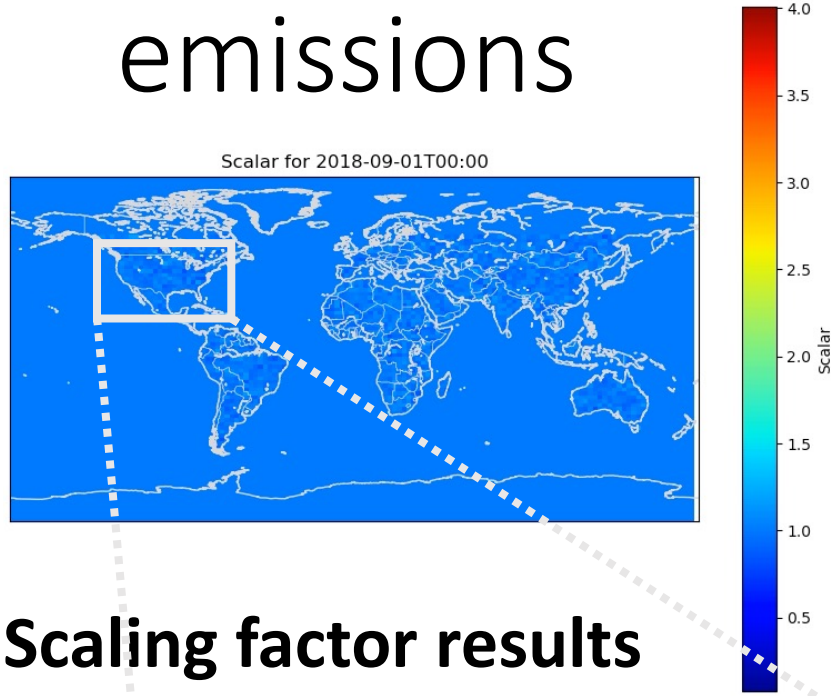
Model state → CH_4

Emissions → CH_4 scaling factors

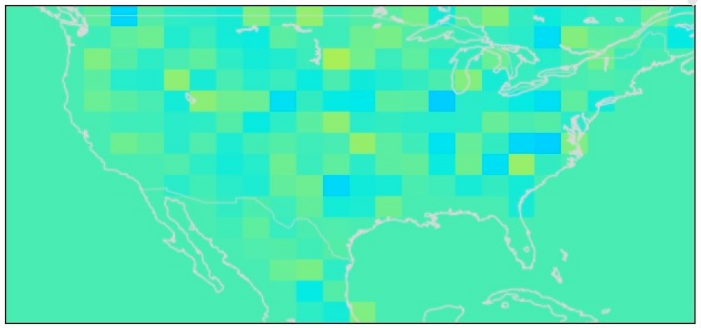
CHEEREIO

Simultaneously optimized CH_4 scaling factors and concentrations

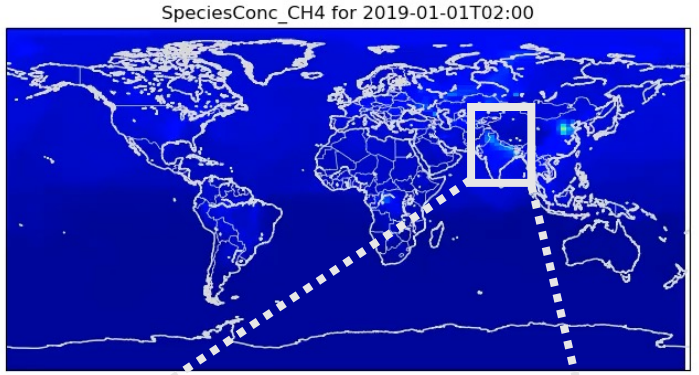
Optimizing methane concentrations and emissions

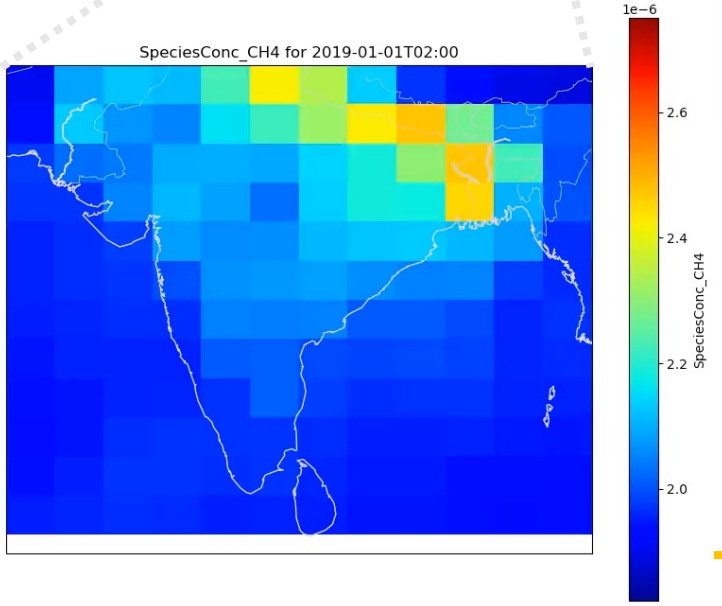


Scaling factor results



Concentration results





Weekly assimilation update.

This doesn't work yet, but we're getting close!

Summary

- CHEEREIO makes it very easy to implement LETKF (ensemble-based) chemical data assimilation with GEOS-Chem with minimal code
- Implemented as a shell that wraps around GEOS-Chem; use whichever version or simulation type that you'd like, including your own customizations, with **no code modifications**.
- Lots of user support tools, including detailed documentation at https://cheereio.readthedocs.io
- Code is open source: https://github.com/drewpendergrass/CHEEREIO
  - *Feel free to download and play with it, but at your own risk!*
- Email me if you want more info, have a development idea, or if you want to know when CHEEREIO is released: **pendergrass@g.harvard.edu**

Background: minimum ensemble values for weekly 2x2.5 global TROPOMI methane assimilation