# Science Workflows Using Kamodo (AGU 2022)

Rebecca Ringuette[1], Lutz Rastaetter[1], Darren De Zeeuw[1], Katherine Garcia-Sage[1], and Oliver Gerland[1]

[1]Affiliation not available

December 27, 2022

# SH42E-2337: Science Workflows using Kamodo


Rebecca Ringuette[1, 2], Lutz Rastaetter[2], Darren DeZeeuw[2,3], Katherine Garcia-Sage[2] and Oliver Gerland[4].

[1]*ADNET Systems Inc., 6720B Rockledge Dr., Suite 504, Bethesda, MD, USA*

[2]*Community Coordinated Modeling Center, NASA Goddard Space Flight Center, Greenbelt, MD, USA*

[3]*Catholic University of America, 620 Michigan Ave., N.E., Washington, DC, USA*

[4]*Ensemble Government Services LLC, Hyattsville, MD, USA*


## Abstract:


**Kamodo** is a powerful python software package based on data functionalization. Once a given data set is functionalized, a large variety of capabilities are easily accessible in Kamodo, including unit conversions, custom analysis via function composition, interactive publication quality visualizations, and LaTeX encoding. The entirety of capabilities available in Kamodo are easily applied to both simulated and observed data across the multiple domains of Heliophysics and even in other disciplines. This work includes a variety of science workflows using Kamodo in combination with other resources, including with other python software packages, that expand the utility of Kamodo even further. These workflows include model-data comparisons, ensemble modeling examples, satellite mission planning examples, and other applications, all of which are freely available on CCMC's Kamodo Github page for the community to adapt to their own uses (https://github.com/nasa/Kamodo). We invite the community to use these workflows and to contribute their own to share.


## Summary:

Three science workflows using Kamodo are presented here:

1. The **Model Data Comparison and Ensemble Modeling** workflow is two workflows combined. The first part performs a flythrough of a given trajectory through three different model outputs in the ITM domain and performs a simple ensemble analysis. The second part shows how to retrieve and functionalize observational data using SPDF's HAPI server and compare it to the simulated results.
2. The **Model-Data Comparison Workflow using Kamodo and pysat** shows how to use Kamodo with pysat to compare model output with cleaned observational data. In this workflow, pysat is used to retrieve and clean the data using an internal cleaning routine. This cleaned data is then easily functionalized and compared to the simulated data from the TIE-GCM model using Kamodo.
3. The **Reconstruction Demo** gives a brief tutorial on how to use the satellite constellation planning tool to compare the simulated observation of a satellite constellation to the modeled prediction. The results of this comparison guide the user in deciding how many and what arrangement of satellites is needed to answer a given science question.

These workflows are examples of how Kamodo can be used to greatly simplify the currently complex process required to use heliophysics model outputs.

## Model Data Comparison and Ensemble Modeling Examples

One can easily compare results from multiple models by simply running the chosen flythrough function for the same trajectory and times for different model outputs. Blocks 1-4 show this process using the RealFlight function. You can easily plug in other flythrough functions by copying the code from the chosen function's notebook into block 1, and other models and variables. Feel free to plug in other options and play! Use the START_HERE notebook to walk you through how to choose the best model and model datasets available on your machine. See the InputOutputDemo notebook for more information on what can be done once the results are functionalized. The relevant data is then retrieved via CDAWeb in block 5 and used in function composition to compare with the average of the model data results.

**Fly a real satellite trajectory through three different model outputs and functionalize the results.**

```
# Run the flythrough for the desired models and variable(s).
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
import datetime as dt

# Set input values for RealFlight function call.
ctipe_file_dir = 'D:/Kamodo_Data/CTIPe/Storm_201303/'
gitm_file_dir = 'D:/GITM/Storm_201303/'
waccmx_file_dir = 'D:/WACCMX/storm_201303/'
start_utcts = dt.datetime(2013, 3, 16, 0).replace(tzinfo=dt.timezone.utc).timestamp()
end_utcts = dt.datetime(2013, 3, 17, 0).replace(tzinfo=dt.timezone.utc).timestamp()-1
variables = ['T_i']  # one or more variable names to retrieve
coord_sys = 'GEO'  # requested cartesian system for satellite positions

# Fly the cnofs trajectory through the simulated data chosen. See SSCWeb for the satellite names.
results_ctipe = SF.RealFlight('cnofs', start_utcts, end_utcts, 'CTIPe', ctipe_file_dir,
                              variables, coord_sys)
results_gitm = SF.RealFlight('cnofs', start_utcts, end_utcts, 'GITM', gitm_file_dir,
                             variables, coord_sys)
results_waccmx = SF.RealFlight('cnofs', start_utcts, end_utcts, 'WACCMX', waccmx_file_dir,
                               variables, coord_sys)
```

```
Attribute/Key names of return dictionary: dict_keys(['sat_time', 'c1', 'c2', 'c3'])
15 times are not in model output files and are excluded from the flythrough.
Time slice index 14 added from file.
```

```
# Functionalize the data.
kamodo_object = SF.O.Functionalize_TimeSeries(results_ctipe['utc_time'], 'CTIPeT_i',
                                              'K', results_ctipe['T_i'])
kamodo_object = SF.O.Functionalize_TimeSeries(results_gitm['utc_time'], 'GITMT_i',
                                              'K', results_gitm['T_i'], kamodo_object)
kamodo_object = SF.O.Functionalize_TimeSeries(results_waccmx['utc_time'], 'WACCMXT_i',
                                              'K', results_waccmx['T_i'], kamodo_object)

kamodo_object
```

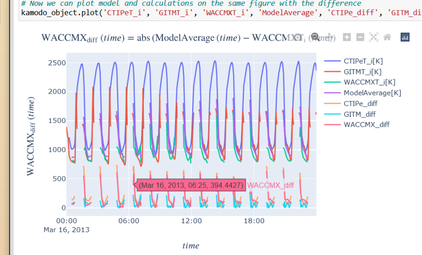
$$\mathrm{CTIPeT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{GITMT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{WACCMXT_i}\,(time)[K] = \lambda(time)$$

**Next, perform some ensemble modeling analysis and plot the results.**

```
# Perform a simple ensembling analysis as an example.
kamodo_object['ModelAverage[K]'] = '(CTIPeT_i + GITMT_i + WACCMXT_i)/3.'
kamodo_object['CTIPe_diff'] = 'abs(ModelAverage - CTIPeT_i)'
kamodo_object['GITM_diff'] = 'abs(ModelAverage - GITMT_i)'
kamodo_object['WACCMX_diff'] = 'abs(ModelAverage - WACCMXT_i)'
kamodo_object
```

$$\mathrm{CTIPeT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{GITMT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{WACCMXT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{ModelAverage}\,(time)[K] = \frac{\mathrm{CTIPeT_i}\,(time)}{3} + \frac{\mathrm{GITMT_i}\,(time)}{3} + \frac{\mathrm{WACCMXT_i}\,(time)}{3}$$
$$\mathrm{CTIPe_{diff}}\,(time) = \mathrm{abs}\,(-\,\mathrm{CTIPeT_i}\,(time) + \mathrm{ModelAverage}\,(time))$$
$$\mathrm{GITM_{diff}}\,(time) = \mathrm{abs}\,(-\,\mathrm{GITMT_i}\,(time) + \mathrm{ModelAverage}\,(time))$$
$$\mathrm{WACCMX_{diff}}\,(time) = \mathrm{abs}\,(\mathrm{ModelAverage}\,(time) - \mathrm{WACCMXT_i}\,(time))$$

```
# Now we can plot model and calculations on the same figure with the difference
kamodo_object.plot('CTIPeT_i', 'GITMT_i', 'WACCMXT_i', 'ModelAverage', 'CTIPe_diff', 'GITM_diff', 'WACCMX_diff')
```

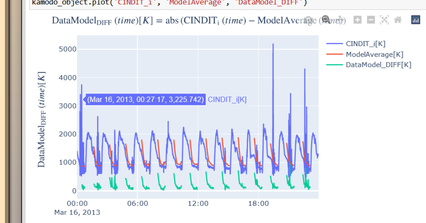



**Now retrieve some relevant observational data and easily perform model-data comparisons.**

```
# Get CINDI data through HAPI
# HAPI sometimes complains. If that happens, just run the notebook again.
from kamodo_ccmc.readers.hapi import HAPI

server = 'https://cdaweb.gsfc.nasa.gov/hapi'
dataset = 'CNOFS_CINDI_IVM_500MS'
parameters = 'ionTemperature'
start = dt.datetime.utcfromtimestamp(start_utcts).replace(tzinfo=dt.timezone.utc).isoformat('T')
stop = dt.datetime.utcfromtimestamp(end_utcts).replace(tzinfo=dt.timezone.utc).isoformat('T')
hapiCDA = HAPI(server, dataset, parameters, start, stop)

# Add the new time series into the Kamodo object and analyze
kamodo_object = SF.O.Functionalize_TimeSeries(hapiCDA.tsarray, 'CINDIT_i',
                                              'K', hapiCDA.ionTemperature(),
                                              kamodo_object=kamodo_object)

kamodo_object['DataModel_DIFF[K]'] = 'abs(CINDIT_i - ModelAverage)'
kamodo_object
```

$$\mathrm{CTIPeT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{GITMT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{WACCMXT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{ModelAverage}\,(time)[K] = \frac{\mathrm{CTIPeT_i}\,(time)}{3} + \frac{\mathrm{GITMT_i}\,(time)}{3} + \frac{\mathrm{WACCMXT_i}\,(time)}{3}$$
$$\mathrm{CTIPe_{diff}}\,(time) = \mathrm{abs}\,(-\,\mathrm{CTIPeT_i}\,(time) + \mathrm{ModelAverage}\,(time))$$
$$\mathrm{GITM_{diff}}\,(time) = \mathrm{abs}\,(-\,\mathrm{GITMT_i}\,(time) + \mathrm{ModelAverage}\,(time))$$
$$\mathrm{WACCMX_{diff}}\,(time) = \mathrm{abs}\,(\mathrm{ModelAverage}\,(time) - \mathrm{WACCMXT_i}\,(time))$$
$$\mathrm{CINDIT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{DataModel_{DIFF}}\,(time)[K] = \mathrm{abs}\,(\mathrm{CINDIT_i}\,(time) - \mathrm{ModelAverage}\,(time))$$

```
# Now we can plot model and data on the same figure with the difference.
# Kamodo does not add data where the model data is not defined.
kamodo_object.plot('CINDIT_i', 'ModelAverage', 'DataModel_DIFF')
```




## Model-Data Comparison Workflow using Kamodo and Pysat

This workflow uses Kamodo to compare simulated in-situ values with observational values along a satellite trajectory.

### Calculating the simulated values along a trajectory

The following block retrieves a real satellite trajectory from SSCWeb and calculates the simulated values of the requested variable(s) for each trajectory location. The code snippet below works for a variety of model names, satellites, time ranges, variables, and coordinate systems, so feel free to play. The output of the RealFlight function is functionalized with a single line at the bottom of the block. The output of kamodo_object shows the LaTeX representation of the variable(s) requested in the RealFlight function call.

```
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
import datetime as dt

# Set input values for RealFlight function call.
model = 'TIEGCM'
file_dir = 'D:/Kamodo_Data/TIEGCM/Uriel_Ramirez_012517_IT_1/'
dataset = 'cnofs'
start_utcts = dt.datetime(2015, 3, 18, 0).replace(tzinfo=dt.timezone.utc).timestamp()
end_utcts = dt.datetime(2015, 3, 21, 0).replace(tzinfo=dt.timezone.utc).timestamp()-1
variable_list = ['T_i'] # list of desired variable names
coord_type = 'GEO'  # GEO cartesian coordinates for the trajectory.
results = SF.RealFlight(dataset, start_utcts, end_utcts, model, file_dir,
                        variable_list, coord_type)

# Functionalize the results
kamodo_object = SF.O.Functionalize_SFResults(model,results)
kamodo_object
```

```
Attribute/Key names of return dictionary: dict_keys(['sat_time', 'c1', 'c2', 'c3'])
21 times are not in model output files and are excluded from the flythrough.
Some requested variables are not available:  ['H_ilev']
Retrieving the H_ilev1 variable instead.
Inverting the pressure level grid. Please wait...
done.
{'T_i': 'K', 'utc_time': 's', 'net_idx': '', 'c1': 'R_E', 'c2': 'R_E', 'c3': 'R_E'}
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$

### Downloading and cleaning the observational data

The following code block uses pysat to download and clean the observational data. The satellite is chosen to be the same satellite as in the above block so the same trajectory is used. The instrument is chosen to correspond to the same variable as in the simulated data above for a proper comparison. Please feel free to change the satellite and variable names in parallel with any changes in the above code block. Keep in mind that different satellites are stored in different sections of pysat, which may require installing additional portions of pysat.

```
# Get CINDI ion temperature data using pysat.
import pysat  # pip install pysat
import pysatNASA  # pip install pysatNASA

pysat_datadir = 'C:/Users/rringuet/Kamodo_Data/pysat_data/'
# pysat.params['data_dirs'] = pysat_datadir  # Only needed the first time.
pysat.utils.registry.register(['pysatNASA.instruments.cnofs_ivm'])
ivm = pysat.Instrument('cnofs', 'ivm', inst_id='', clean_level='clean')
start_download_date = dt.datetime.utcfromtimestamp(start_utcts)
stop_download_date = dt.datetime.utcfromtimestamp(end_utcts)
# ivm.download(start_download_date, stop_download_date)  # already on machine
ivm.load(date=start_download_date, end_date=stop_download_date)
```

### Functionalizing the observational data

Next, we must functionalize the observational data to access the capabilities of Kamodo for the comparison. Pysat returns data in pandas dataframes, so a few lines of code are necessary to retrieve the utc timestamps. Once that is completed, the observational time series is functionalized in a single command (kamodo_object = ...). With both the simulated and observational data in the same Kamodo object, various Kamodo capabilities can be used to analyze the datasets, including function composition as shown at the end of the block. If a different measurement is used in the previous blocks, you will need to use pysat's features to retrieve the proper unit and variable name for the observational data.

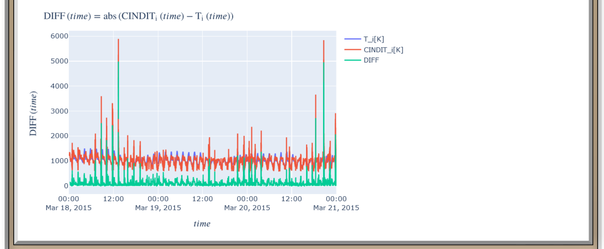
```
# Convert pandas timestamps into utc timestamps
import numpy as np
utc_time = ivm.index.values.astype(np.int64)/1e9

# Add to kamodo object.
kamodo_object = SF.O.Functionalize_TimeSeries(utc_time, 'CINDIT_i', 'K',
                                              ivm.data['ionTemperature'].values,
                                              kamodo_object)

kamodo_object['DIFF'] = 'abs(CINDIT_i - T_i)'
kamodo_object
```

$$\mathrm{T_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{CINDIT_i}\,(time)[K] = \lambda(time)$$
$$\mathrm{DIFF}\,(time) = \mathrm{abs}\,(\mathrm{CINDIT_i}\,(time) - \mathrm{T_i}\,(time))$$

### Visualization

Now that the simulated and observational data have been functionalized, we can easily plot any functions desired with a single command. The figure can be saved in full interactive form. Please see the Plotly documentation for more details on this feature. https://plotly.com/python-api-reference/index.html

```
# Now we can plot model and data on the same figure with the difference
kamodo_object.plot('T_i','CINDIT_i','DIFF')
```




## Links to our GitHub Repositories:

**Get the CCMC's Kamodo!**

**Get the core Kamodo!**

## Demo notebook for Kamodo's Reconstruction: Spherical Coordinates

### The short version

This notebook tutorial shows how to fly a constellation of satellites through model data as a virtual reality, focusing on spherical coordinate examples. Trajectories can be obtained either from the test GDC trajectory file or the flythrough trajectory functions as shown here. See the Trajectory_Coord_Plots demo notebook for examples on flythrough trajectory options, and the START_HERE notebook for more information on choosing models and variables.

```
# For a given model, find out what time range is covered by the data in a given directory.
from kamodo_ccmc.flythrough import model_wrapper as MW
model, file_dir = 'GITM', 'D:/GITM/jasoon_shim_071418_IT_1_tenth/'  # change to match your machine
times = MW.File_Times(model, file_dir)
times[0].timestamp(), times[1].timestamp()
# This function also automatically performs any data preparation needed.
```

```
UTC time ranges
----------------------------------------
Start Date: 2015-03-17  Time: 00:00:00
End Date: 2015-03-19  Time: 01:48:59

(1426550400.0, 1426729739.996338)
```

```
# Import function to retrieve the grace1 trajectory from the SSCWeb.
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
# Typical coordinates possible through SSCWeb are GEO, GSE, SM, and GSM (all cartesian and in R_E).
input_coord = 'GEO'
traj_dict, coord_type = SF.SatelliteTrajectory('grace1', times[0].timestamp(), times[1].timestamp(), coord_type=input_coord)
```

```
Attribute/Key names of return dictionary: dict_keys(['sat_time', 'c1', 'c2', 'c3'])
```

```
# Convert coordinates to system desired for reconstruction to take place in.
# The trajectories were retrieved in GEO cartesian, and are converted to GEO spherical below.
from kamodo_ccmc.flythrough.utils import ConvertCoord
c1, c2, c3, units = ConvertCoord(traj_dict['sat_time'], traj_dict['c1'], traj_dict['c2'], traj_dict['c3'],
                                 input_coord, 'car', input_coord, 'sph')
print(c1.min(), c1.max(), c2.min(), c2.max(), c3.min(), c3.max())
```

```
-179.9064985622182 179.91158102702934 -88.98756139868595 88.99006780797983 1.0599069924370896 1.0644682426787557
```

```
# Choose inputs.
variable_name = 'rho_n'  # from chosen files above
recon_dimensions = 'c1c2'  # Longitude vs Latitude reconstruction for spherical coordinates
recon_option = 'Unmod_AvgOSlice'
# fly the given trajectory through the data unmodified, then fly the reconstructed coordinate grid through the data
# after taking the average of t and height.
lon_offsets = [0., 30., 60., 90., 120., 150.]  # 6 satellites equally spaced in longitude
# Choose the grid resolution of reconstruction. The finer the resolution, the longer the program takes to run and
#   the more 'holes' you will see in the reconstructed plot. Physically, these should be set to the instrument's
#   field of view in the units of the input coordinate system (e.g. degrees for longitude and latitude, seconds for
#   time, etc).
dx, dy = 4., 2.  # Since recon_dimensions='c1c2', dx is resolution in longitude, and dy is the resolution in latitude.
d1, d2 = 1800., 0.001  # time (in s) and height (in R_E) resolution of sampling for averaging

# Run the reconstruction.
recon = RECON(model, variable_name, file_dir, traj_dict['sat_time'], c1, c2, c3,
              input_coord, 'sph', recon_option, recon_dimensions, c1_offsets=lon_offsets, dx=dx, dy=dy, d1=d1, d2=d2)
```

```
Grid flythrough completed in 1015.87044 s.
Reconstruction program complete in 1030.74607 s.
```

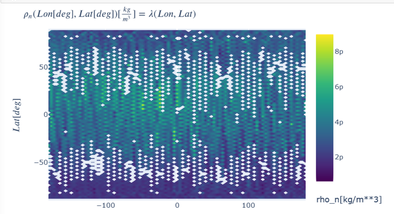
```
# The output of the function is a Kamodo object with all of the default features described in documentation.
# rho_n is the reconstructed data from the constellation flythrough. rho_n_model is the data from the model in the
#   method chosen. PercentDiff is the percent difference between the two, calculated using
#   PercentDiff = (rho_n_model - rho_n)/rho_n_model*100.
recon
```

$$\rho_n(Lon[deg], Lat[deg])[\frac{kg}{m^3}] = \lambda(Lon, Lat)$$

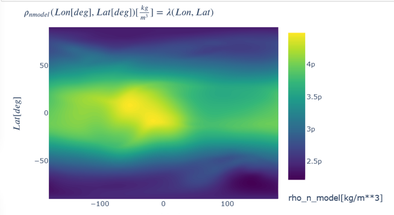
$$\rho_{nmodel}(Lon[deg], Lat[deg])[\frac{kg}{m^3}] = \lambda(Lon, Lat)$$

$$\mathrm{PercentDiff}(Lon[deg], Lat[deg]) = \lambda(Lon, Lat)$$

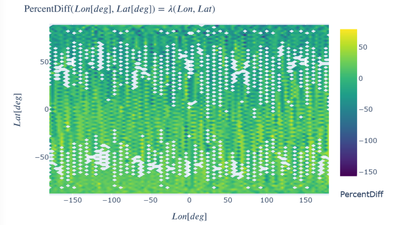
```
fig = recon.plot(rho_n=dict(lon=recon.x, lat=recon.y))
fig.update_traces(colorscale="Viridis", ncontours=200,
                  contours=dict(coloring="fill",showlines=False))
fig
```




```
fig = recon.plot(rho_n_model=dict(lon=recon.x, lat=recon.y))
fig.update_traces(colorscale="Viridis", ncontours=200,
                  contours=dict(coloring="fill",showlines=False))
fig
```




```
# Show the percent difference between the two. A percent difference of zero is an exact match.
fig = recon.plot(PercentDiff=dict(lon=recon.x, lat=recon.y))
fig.update_traces(colorscale="Viridis", ncontours=200,
                  contours=dict(coloring="fill",showlines=False))
fig
```




## Our Team:

### CCMC Staff:

https://ccmc.gsfc.nasa.gov/staff/

*Rebecca Ringuette*: Model interfaces, metadata, flythrough and other CCMC capabilities.

*Lutz Rastaetter*: Internal cross-language interfaces, specialized interpolators, CCMC-Vis, team management.

*Darren De Zeeuw*: GitHub management, visualization, metadata.

*Katherine Garcia-Sage*: Orbit propagation and satellite reconstruction studies, external affairs, GDC support

### Ensemble Government Services partners:

https://www.ensembleconsultancy.com/government-services

*Oliver Gerland and company*: Core Kamodo capabilities, expert bug squashers.


### Acknowledgements:

Russell Stoneback for assistance with the Kamodo-pysat notebook.


## Related Materials:

### Related Posters and Papers:

- SH42E-2338: Magnetic Mapping in the Inner Magnetosphere using Kamodo
- SA32D-1694: Enhanced Visualization using Kamodo for CCMC ITM Instant Runs
- Kamodo's Satellite Constellation Mission Planning Tool SM25C-2002
- Developing an Executable Paper With the Python in Heliophysics Community. Preprint DOI: 10.1002/essoar.10510006.1 Accepted by *Frontiers in Astronomy and Space Science: Space Physics.*

### Reference DOIs:

- Kamodo (core): 10.21105/joss.04053
- CCMC's Kamodo Flythrough: 10.3389/fspas.2022.1005977
- CCMC's Kamodo Model Readers: under review by ASR.
- HAPI: 10.1029/2021JA029534
- Pysat: 10.1029/2018JA025297

### Note:

If you find an issue with the software, please report it on our GitHub. For collaboration, please email **Rebecca.ringuette@nasa.gov**.