SA32D-1694: Using Kamodo for CCMC ITM Output and Beyond

Darren DeZeeuw[1,2], Mark Moussa[1], Rebecca Ringuette[3, 1], Lutz Rastaetter[1], Rita Owens[1] and Chiu Wiegand[1].

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

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

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

Updated Abstract:

Kamodo is an official NASA open source python software package that functionalizes diverse datasets from models and observations in a consistent way, enabling advanced scientific analysis and visualization with simplistic syntax. Here we demonstrate this ability using several ITM models available through the Community Coordinated Modeling Center (CCMC). Users can now interact directly with model outputs, and satellites can be virtually flown through model output to allow many types of model/model and data/model comparisons. We will also provide information about significant updates and improvements to Kamodo and future plans.

The core of Kamodo is supported by Ensemble Government Services and is available here:

https://github.com/EnsembleGovServices/kamodo-core

The CCMC Kamodo package supports many model and data sources and adds unique functionality. Many sample workflows are freely available on CCMC's Kamodo Github page for the community to adapt to their own uses. We invite the community to use these workflows and to contribute their own to share. The CCMC Kamodo is available here:

https://github.com/nasa/Kamodo

Related Materials:

Related Posters and Papers:

- SH42E-2338: Magnetic Mapping in the Inner Magnetosphere using Kamodo
- SH42E-2337: Science Workflows using Kamodo
- SM25C-2002: Kamodo's Satellite Constellation Mission Planning Tool
- 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 Advances in Space Research.

Note:

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

Get CCMC's Kamodo here!

Standard Kamodo Use:

Show a list of models currently available

```
import kamodo_ccmc.flythrough.model_wrapper as MW

MW.Choose_Model('')
```

```
Possible models are:
CTIPe: Coupled Thermosphere Ionosphere Plasmasphere Electrodynamics Model
GITM: Global Ionosphere Thermosphere Model
IRI: International Reference Ionosphere Model
SWMF_IE: Space Weather Modeling Framework - Ionosphere and Electrodynamics outputs
SWMF_GM: Space Weather Modeling Framework - Global Magnetosphere outputs
TIEGCM: Thermosphere Ionosphere Electrodynamics General Circulation Model
OpenGGCM_GM: The Open Geospace General Circulation Model - Global Magnetosphere outputs only
AMGeO: Assimilative Mapping of Geospace Observations
SuperDARN_uni: SuperDARN uniform grid output
SuperDARN_equ: SuperDARN equal area grid output
ADELPHI: Coming soon
WACCMX: Whole Atmosphere Community Climate Model With Thermosphere and Ionosphere Extension
WAMIPE: The coupled Whole Atmosphere Model - Ionosphere Plasmasphere Model
DTM: The Drag Temperature Model
```

New

Retrieve the model reader and load the requested variable.

```
reader = MW.Model_Reader('CTIPe')
kamodo_object_ctipe = reader(ctipe_file_dir, variables_requested=['T_i'])
kamodo_object_ctipe
```

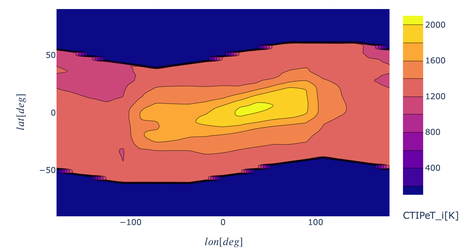
$T_i\ (\vec{r}_{GDZ;phkD})[K] = \lambda(\vec{r}_{GDZ;phkD})$

$T_{ijk}(time[hr], lon[deg], lat[deg], height[km])[K] = \lambda(time, lon, lat, height)$

Default Kamodo plotting for a slice at given time, height

```
kamodo_object.plot('CTIPeT_i', plot_partial={'CTIPeT_i': {'time': 12., 'height': 500.}})
```

$\mathrm{CTIPeT_i}(lon[deg], lat[deg])[K] = \mathrm{CTIPeT_i}(time[hr], lon[deg], lat[deg], height[km])[K]$

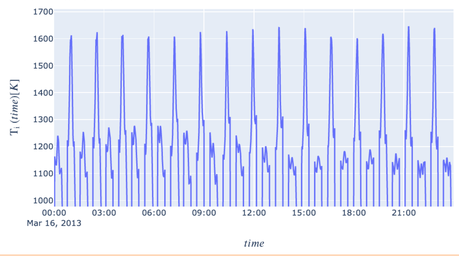


Fly a satellite through the model output and plot it

```
from kamodo_ccmc.flythrough import SatelliteFlythrough as SF
model = 'CTIPe'
variable_list = ['T_i']
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
traj_dict, new_coord_sys = SF.SatelliteTrajectory('grace1', start_utcts, end_utcts,
                                                  coord_type='GSE')
# Run ModelFlythrough with grace1 trajectory from SSCWeb
results = SF.ModelFlythrough('CTIPe', ctipe_file_dir, ['T_i'], traj_dict['sat_time'],
                             traj_dict['c1'], traj_dict['c2'], traj_dict['c3'], new_coord_sys)
kamodo_object_fly = SF.Functionalize_SFResults(model, results)
```

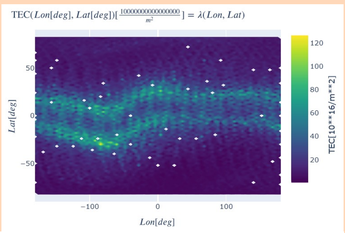
```
kamodo_object_fly.plot('T_i')
```

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



Satellite Constellation Mission Planning Tool: Irregular Constellations

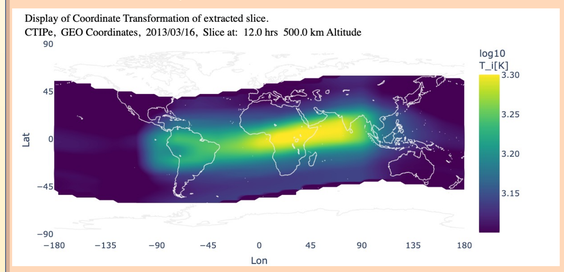
Kamodo has developed a tool to assist satellite constellation mission studies, called the 'reconstruction' tool, particularly in support of the GDC (see SM25C-2002). The syntax for this tool is the same regardless of the model and can be used for any model supported in Kamodo.



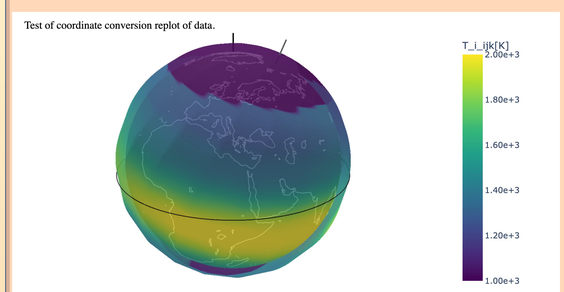
The reconstructed TEC (total electron content) for the DMSPF15, DMSPF16, DMSPF17, and DMSPF18. The TEC data at each trajectory location is interpolated from WAM-IPE data ranging from March 16-20 of 2013 during a geomagnetic storm. The data were reconstructed on a longitude-latitude grid with a grid resolution of 4° by 2° for all time values. Empty grid locations on the plot indicate a longitude-latitude location that was not sampled by the constellation.

Expanded Custom Visualizations:

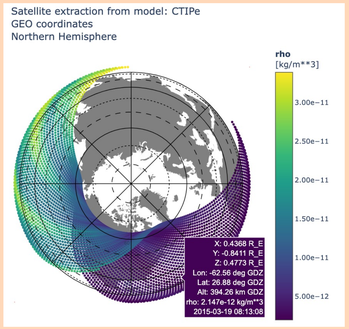
The custom plot below is the same data as the default ion temperature plot in the column to the left, but with several enhancements. It has more descriptive plot labeling, land/water boundaries shown, plotted in log scale with custom contour ranges, and can be shown in a many different coordinate systems.



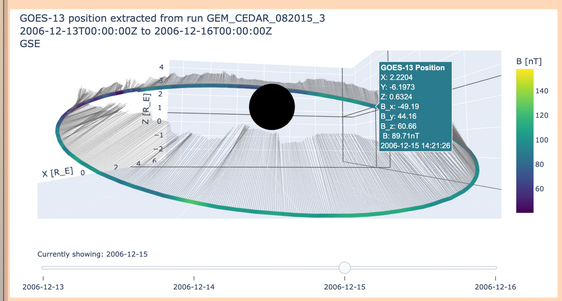
The next custom plot is again the same data, but now in 3D and in GSE coordinates rather than the model default of GDZ. The GSE poles are shown as well as geographic poles.



A virtual satellite was flown through the CTIPe model output and the density values at the satellite positions were extracted. The output can be shown in 1D, 2D, 3D, or 4D plots in several coordinate systems. Shown is a 2D polar plot in geographic coordinates. The hover box shows other useful information at the desired location.



Below is a custom visualization of the magnetic field vector and magnitude extracted from a MHD simulation at GOES-13 positions over a four day period. At the bottom of the plot is a time slider to select the day to display.



Note that all of the figures shown are using Plotly and are full interactive. They can be zoomed, rotated, and saved as desired.

Model Reader Redesign:

The Kamodo model readers have recently been redesigned to functionalize the entire dataset in a file directory instead of only a piece if the data that is predetermined to fit into memory. In place of the previous approach, we have implemented a **lazy interpolation method** for all datasets that have a spatial component.

In the lazy interpolation scheme, the time slices of the data required for the interpolation are **loaded into memory on demand**. This speeds up the kamodo object creation (*kamodo_object = reader(file_dir)*), but increases the interpolation time by the amount of time it takes to read in the data. The increased execution time is compensated for by streamlined logic and remains **two orders of magnitude faster** than the previous dask + xarray solution.

Three lazy interpolation methods are currently supported by the new model reader infrastructure. These three methods are motivated by the common file structures encountered so far: (1) files that have data for the given variable for one timestep and fit into memory ("**Time Step**" in the table below), (2) file that have data for the given variable for multiple timesteps and fit into memory ("**Multiple Time Steps**"), and (3) files that have data for the given variable for multiple timesteps and do not fit into memory ("**Time Step (large file)**"). The table below shows which model outputs use which interpolation method, include a specialized interpolator, or have time series data.

Kamodo Model Reader Interpolation Table

Model	Interpolation Method
AMGeO*	Multiple Time Steps
CTIPe^	Multiple Time Steps
DTM	Multiple Time Steps
GITM	Time Step
IRI	Multiple Time Steps
OpenGGCM (mag.)	Time Step
SuperDARN (uniform)*	Time Step#
SuperDARN (equal area)*	Time Step#
SWMF (iono. electro.)	Time Step
SWMF (mag.)	Time Step#
TIE-GCM^	Time Step or Multiple Time Steps
WACCM-X^	Multiple Time Steps or Time Step (large file)
WAM-IPE^	Time Step

*Model reader contains time series data, which is functionalized by interpolating through the entire variable dataset at once.

#A specialized interpolator is incorporated into this model reader. For SWMF (mag.), that interpolator is called from C.

^Variables in this model reader require an inversion of the model-specific pressure level grid.

Another important feature in the redesigned model readers is the **easy insertion of a custom interpolator** into a model reader script as shown below for all interpolation methods except the zero interpolation method. This important feature is necessary for any coordinate grid that changes with time or is irregular. This structure is being used to implement a custom interpolator written in C for the SWMF magnetosphere output (see SH42E-2338).

```
def func(i):
    '''i is the file number.'''
    # get data from file
    file = self.pattern_files[key][i]
    cdf_data = Dataset(file)
    data = array(cdf_data.variables[gvar])
    cdf_data.close()
    return data

# define and register the interpolators
self = RU.Functionalize_Dataset(self, coord_dict, varname,
                                self.variables[varname],
                                gridded_int, coord_str,
                                interp_flag=1, func=func)
```

Normal (GITM model)

```
def func(i):
    '''i is the file/time number.'''
    # get data from file
    file = self.pattern_files[key][i]
    cdf_data = Dataset(file)
    data = array(cdf_data.variables[gvar])
    lat = array(cdf_data.variables['lat'])
    cdf_data.close()
    # data wrangling all done in the file conversion step
    coord_list = [coord_dict['lon']['data'], lat]
    interp = RU.rgiND(coord_list, data, bounds_error=False,
                      fill_value=NaN)
    return interp

# functionalize the variable dataset
self = RU.Functionalize_Dataset(
    self, coord_dict, varname, self.variables[varname],
    gridded_int, coord_str, interp_flag=1, func=func,
    func_default='custom')
```

Custom (SuperDARN model)

Other changes include a faster and more accurate pressure level inversion; simpler, more flexible code structure; and the removal of file directory searches for all except the first execution per file directory (now less expensive to run on the cloud). See our upcoming paper in ASR for more details.

Our Team:

CCMC Staff:

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

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

Rebecca Ringuette: Kamodo model interfaces, metadata, flythrough and other CCMC capabilities.

Lutz Rastaetter: Kamodo internal cross-language interfaces, specialized interpolators, CCMC-Vis, team management.

Chiu Wiegand: CCMC software team lead

Mark Moussa: CCMC software developer for ITM instant runs

Rita Owens: CCMC software developer, model containerization expert.

Acknowledgements:
Katherine Garcia-Sage and Robert Robinson for assistance with the satellite reconstruction tool.

Future Plans:

CCMC Improvements:

- Working to add more models, from ITM to Geospace and eventually out to Solar and Heliosphere models. The next models coming soon are **GAMERA**, **MARBLE**, and **GUMICS**.
- More satellite constellation studies
- Validation study support, neutral density
- Model-defined coordinate conversions via function composition
- Parallelize custom interpolators
- Add a HAPI layer on top of flythrough function
- Develop a line-of-sight calculation tool
- Link Kamodo to SunPy for image model/data comparisons

Open Science:

- Improving code documentation and sample workflows to improve access and usability.

Kamodo-core:

- Cross language interfaces on cloud
- Code free interfaces
- Migrating from Python 3.7 to 3.11
- Exploring commercial applications
- Bug fixes

Get Kamodo-core here!