Kamodo and CCMC-Vis: CCMC Tools for Visualization and Analysis

Rebecca Ringuette[1, 2], Lutz Rastaetter[2], Darren DeZeeuw[2], Asher Pembroke[3,4], Katherine Garcia-Sage[2], Oliver Gerland[3], Michael Contreras[3], and Robert Robinson[5, 2].

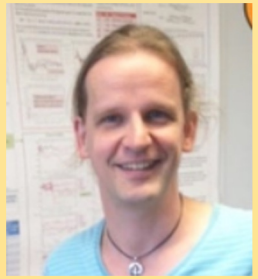
[1]*ADNET Systems, Inc.*, [2]*NASA Goddard Space Flight Center*, [3]*Ensemble Consultancy*, [4]*Satoshi Energy*, [5]*Catholic University of America*

Kamodo-Core

CCMC-Kamodo

CCMC 2022 Workshop

The Kamodo Team



CCMC 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.

Ensemble Consultancy partners:

- *Asher Pembroke & Oliver Gerland*: Core Kamodo capabilities, expert bug squashers.

CCMC-Kamodo

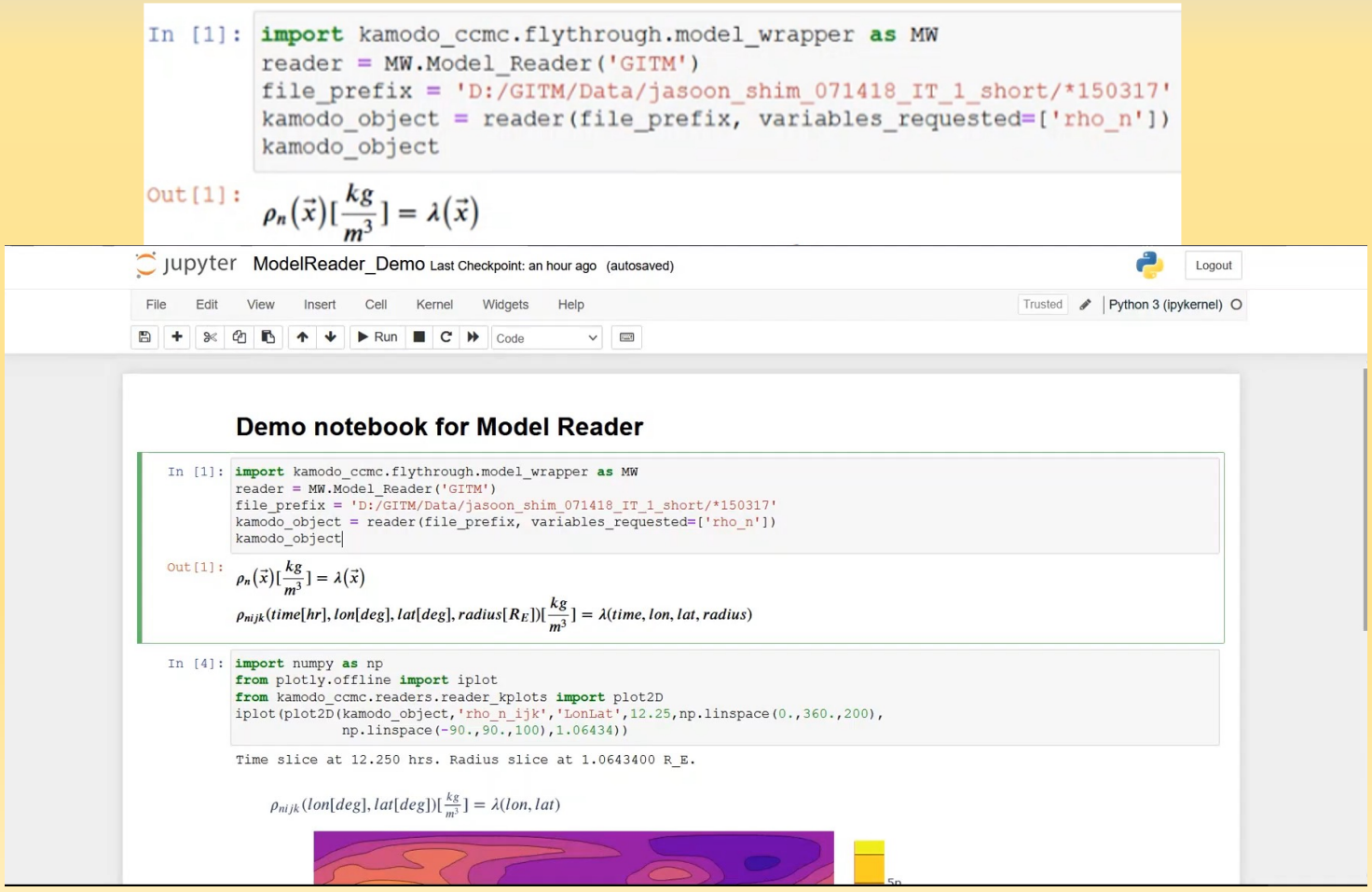
Kamodo-Core

Outline

- *Rebecca Ringuette*: Overview, model interfaces, metadata.
- *Darren de Zeeuw*: Data-model comparison using Kamodo's flythrough, interactive visualization.
- *Lutz Rastaetter*: Cross-language interfaces, specialized interpolators, CCMC-Vis.
- *Asher Pembroke*: Calling Kamodo from other languages, Code-free and reduced code interfaces.

Kamodo: Simplifying Model Data Access at CCMC

- **Model Reader**: *Model-agnostic* direct access to functionalized model data (currently ITM and Magnetosphere).
- **Flythrough**: Fly a trajectory time-series (real, generated, or user-given) through model data.
- **Constellation Tool**: Reconstruct what a satellite constellation would see using model data as a virtual reality.
- **PyHC Executable Paper:** Demonstration of model-data comparison and interoperability with other packages.



Kamodo: New and Planned Features

- **Specialized Interpolators:** Block-adaptive grid interpolator for SWMF/BATS-R-US and GUMICS
- **Code-free interfaces:** Create code-free workflows with functionalized data.
- **Coming soon:**
  - Link to orbit propagators and TLEs as inputs
  - Expand model and interpolator library
  - Interpolator parallelization
  - Online analysis container with model data
  - Direct interactive access from other languages (e.g. C, C++, Fortran)

BATSRUS/GUMICS block grid interpolator

CFFI compilation script with interface routine definition

```
from cffi import FFI
ffibuilder = FFI()
ffibuilde.cdef("""
typedef int IDL_LONG;
typedef struct {int refinement_level;int parent_ID;int child_count;int child_IDs[8];float
XMIN,XMAX,XCenter,YMIN,YMAX,YCenter,ZMIN,ZMAX,ZCenter;} octree_block;
int find_octree_block(float x, float y, float z, int old_blocknumber, int max_level);
int xyz_ranges(int N, float *x, float *y, float *z, float *XR_BLK, float *YR_BLK, float *ZR_BLK, float *X_BLK, float
*Y_BLK, float *Z_BLK, float *box_range, int positions_in_cell_center);
void   setup_octree(IDL_LONG N_blks_in,
                    float *xr_blk, float *yr_blk, float *zr_blk,
                    IDL_LONG MAX_AMRLEVELS, float *box_range,
                    octree_block *octree_blocklist_in,
                    IDL_LONG N_octree_blocks,
                    IDL_LONG *numparents_at_AMRlevel_in,
                    IDL_LONG *block_at_AMRlevel_in);

float interpolate_amrdata(float xx,float yy,float zz, float *field, IDL_LONG is_new_position);
int interpolate_amrdata_multipos(float *xx,float *yy,float *zz, int npos, float *field, float *output);
""")
ffibuilder.set_source("_interpolate_amrdata",  # name of the output C extension
"""
// always add any public function declaration in the separate ffibuilder.cdef declaration!
    #include <stdlib.h>
    #include "fl.h"
    #include "octree_block.h"
    #include "setup_octree.h"
    int interpolate_amrdata_multipos(float *xx, float *yy, float *zz, int npos,float *data, float *output){
        for (int ipos=0;ipos<npos;ipos++){
            output[ipos]=interpolate_amrdata(xx[ipos],yy[ipos],zz[ipos],data, 1);
        }
        return(0);
    }
""",
sources=['interpolate_amrdata.c','xyz_ranges.c','setup_parent.c','setup_octree.c','interpolate_in_block.c','find_oct
ree_block.c','find_in_block.c','find_block.c'],   # includes additional sources
    libraries=['m'])  # on Unix, link with the math library
if __name__ == "__main__":
    ffibuilder.compile(verbose=True,debug=False)
    #ffibuilder.compile(verbose=True)
    #ffibuilder.compile()
```

Function declarations

Interpolator

C source files and compilation

Kamodo: Demo Notebooks

- Model Reader notebook: https://drive.google.com/file/d/1oknnmk53mwmdQxmq2liQ-Aeustcpv2fR/view?usp=sharing
- Model-Data notebook Part 1: https://github.com/nasa/Kamodo/blob/master/notebooks/Model_Data_Comparison-TIEGCM.ipynb
- Model-Data notebook Part 2:

https://github.com/nasa/Kamodo/blob/master/notebooks/Model_Data_Comparison-ROR.ipynb

Kamodo: Specialized Interpolators and Readers

SWMF/BATSRUS, GUMICS:

- Interpolator for block-adaptive grids written in C is linked to Python via *cffi*.
- Code suitable for:
  - BATSRUS IDL (cell centered) data,
  - GUMICS TecPlot (cell-corner based) or
  - GUMICS DC (cell-centered) data.

Plans / Collaboration Opportunities:

- Extend interpolator to solar-heliosphere applications:
  - spherical coordinates
  - generalized radial coordinate
  - ARMS model (MHD of solar corona)
- Implement parallelized interpolation in 4D (time+space)
  - 3D interpolators arranged in time, run in parallel to assemble time series.

Python interpolator using C interface function and library.

```
#
# obtain number of blocks using the xyz_ranges() function and allocation of array of octree_block structures
(Fig.3)
#
xr_blk=OCTREE_BLOCK_GRID_FFI.new("float[]",N_4) # 2 elemts per block, smallest blocks have 2x2x2=8 positions
yr_blk=OCTREE_BLOCK_GRID_FFI.new("float[]",N_4) # at cell corners (GUMICS), so maximum array size is N/4
zr_blk=OCTREE_BLOCK_GRID_FFI.new("float[]",N_4) # BATSRUS blocks have at least 4x4x4 cell centers
x_blk=OCTREE_BLOCK_GRID_FFI.new("float[]",N_4) # 2 elements per block, smallest blocks have 2x2x2=8 positions
y_blk=OCTREE_BLOCK_GRID_FFI.new("float[]",N_4) # at cell corners (GUMICS), so maximum array size is N/4
z_blk=OCTREE_BLOCK_GRID_FFI.new("float[]",N_4) # BATSRUS blocks have at least 4x4x4 cell centers
box_range=OCTREE_BLOCK_GRID_FFI.new("float[]",6)
N_blks=N_blks=OCTREE_BLOCK_GRID_LIB.xyz_ranges(Ncell,x,y,z,xr_blk,yr_blk,zr_blk,x_blk,y_blk,z_blk,box_range,1)
N_octree=int(N_blks*8/7)
octree_blocklist=OCTREE_BLOCK_GRID_FFI.new("octree_block[]",N_octree)
# Code to complete population of octree_blocklist array used by interpolate_amrdata() is not shown.
# Variable registration:
#   Registration includes creating a Python interpolator wrapper for each quantity.
#   Interpolators are then stored in the Kamodo object along the data and units.
    def register_variable(self, varname, unitss
        """register variables into Kamodo for this model, SWMF-GM"""
         def interpolator(xvec):
              var_data=OCTREE_BLOCK_GRID_FFI.new("float[]",list(self.variables[varname]['data']))
              xvec=np.array(xvec)
              xpos=OCTREE_BLOCK_GRID_FFI.new("float[]",list(xvec[:,0]))
              ypos=OCTREE_BLOCK_GRID_FFI.new("float[]",list(xvec[:,1]))
              zpos=OCTREE_BLOCK_GRID_FFI.new("float[]",list(xvec[:,2]))
              npos=len(list(xvec[:,0]))
              return_data=list(np.zeros(npos,dtype=np.float))
              return_data_ffi=OCTREE_BLOCK_GRID_FFI.new("float[]",return_data)
              python_loop=False
              if python_loop:
                  for i in range(npos):

return_data[i]=OCTREE_BLOCK_GRID_LIB.interpolate_amrdata(xvec[i,0],xvec[i,1],xvec[i,2],var_data,1)
              else:

IS_OK=OCTREE_BLOCK_GRID_LIB.interpolate_amrdata_multipos(xpos,ypos,zpos,npos,var_data,return_data_ffi)
                  return_data[:]=list(return_data_ffi)
              toc=time.time()
              return return_data
        # store the interpolator
        self.variables[varname]['interpolator'] = interpolator
```

Kamodo: Specialized Interpolators and Readers

OpenGGCM magnetosphere and ionosphere electrodynamics:

- compressed files decoded using Fortran library via *f2py*.
  - Conversion performed on CCMC server.
  - Create 4D (space+time) NetCDF files for Kamodo + Flythrough
  - Access to large data files using Xarray.

Opportunities for Collaborations:

- Implement parallelized interpolation in 4D (time+space)
- Split large grids into sub-regions as needed for faster regional analyses (near-Earth, dayside, etc.).
- Use 3D reader/interpolators in parallel to create 4D interpolation.

Using OpenGGCM Fortran library

```
[ kamodo_ccmc/readers/OpenGGCM $] cat readmagfile3d.f

c-----------------------------------------------------------------------
      subroutine read_3d_field(path3d,
     &           threeDfield,var,nx,ny,nz,asciiTime)
c-----------------------------------------------------------------------
!     character(len=40) :: path3d
      character(len=110) :: path3d
      character*80 var
      character*80 asciiTime
      real threeDfield(nx,ny,nz)
      integer fileNo2
      integer nx
      integer ny
      integer nz
      integer it
!f2py intent(in) :: path3d
!f2py intent(in,out) :: threeDfield
!f2py intent(in) :: var
!f2py intent(out) :: nx,ny,nz
!f2py intent(out) :: asciiTime
      fileNo2=11
c     write (0,*) 'opening file: |',path3d,'|'
      open(unit=fileNo2,file=path3d,status='old',action='read',err=222)
      call read_mhd3d(fileNo2,threeDfield,nx,ny,nz,var,asciiTime,it)
      close(fileNo2)
      return
 222  write(0,*) "could not open file"
      return
      end
. . .
```

Signature of entry point routine.

Build readOpenGGCM library with *f2py*:

```
f2py -c -m readOpenGGCM  \
            read_b_grids.f  \
            read_ctim2d.f \
            readmagfile3d.f
```

Specification for *f2py* of inputs and outputs.

Python use:

```
import readOpenGGCM as ropgm
. . .
# read 'bx' from 3D data file (*.3df.??????):
(field3d,nx,ny,nz,timestring) =
   ropgm.read_3d_field(filepath, fieldbuffer, 'bx')
```

Kamodo: Specialized Interpolators and Readers

Collaboration Opportunities:

GUMICS:

- Added support for GUMICS-5 *.dc files in IDL (CCMC online vis.):
  - Grids as large as 15 million cells tested.
  - Possible to read subregions of grid.

GAMERA:

- Model writes multi-timestep magnetosphere HDF-5 output files.
- MPI model generates outputs for separate parts of grid:
  - Parallelized reader and interpolation.
- Specialized Fortran interpolator to be supplied by modelers.

CCMC-Vis: Added Features

User Interface Modernized:

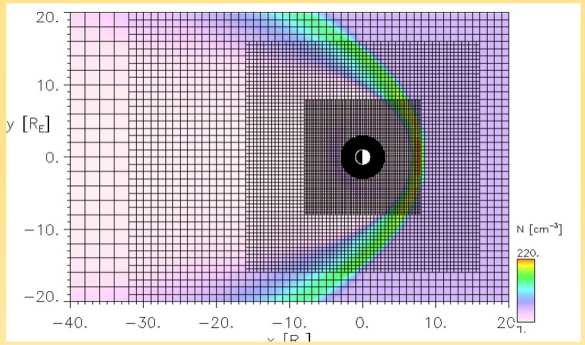
- Only items relevant to Plot Mode selection shown.
- Advanced options on demand only.

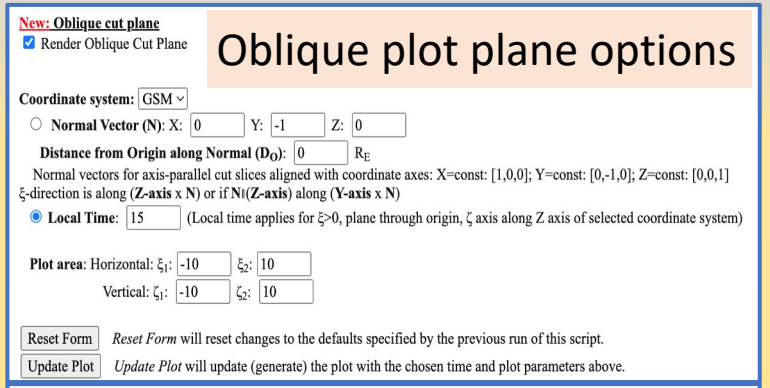
Plot features added:

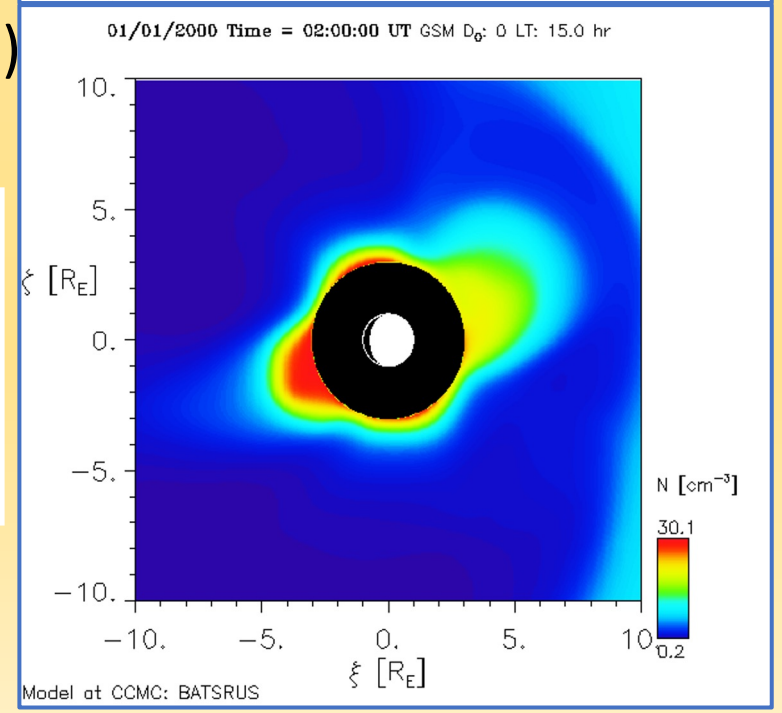
- Oblique slices and coordinate transformation (SM-GSM-GSE)
- Side-by-side plot comparison for multiple runs.
- Choice of color tables.

Models added:

- SH: Euhforia
- GM: HYPERS, GAMERA, GUMICS-5
- IM: IMPTAM, VERB-3D
- IT: PANDOCA, JB, IRI, IGRF (run-on-request runs)
  WACCM-X, Adelphi, AmGEO, SuperDARN

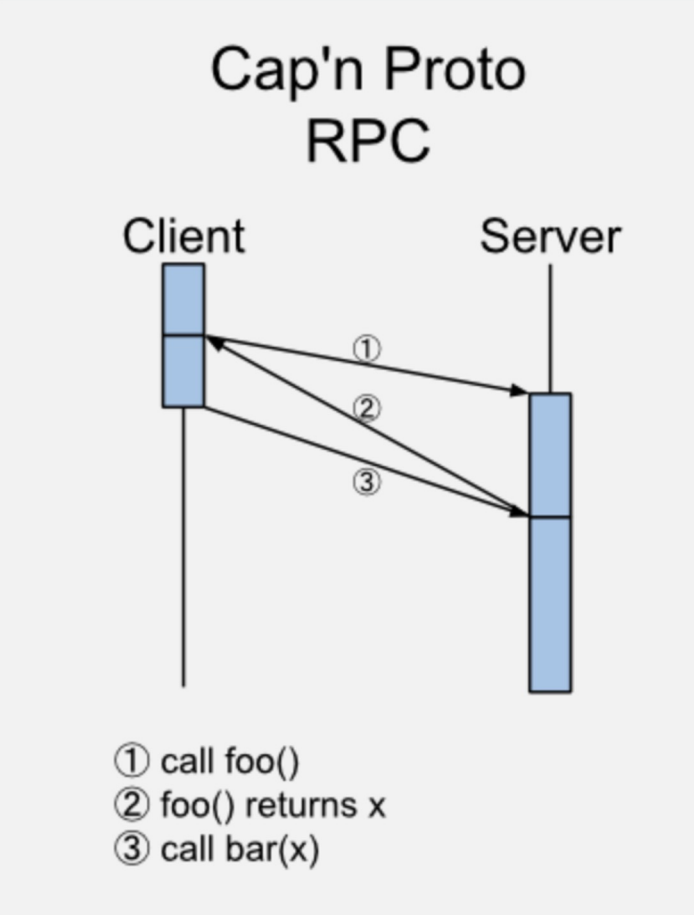






Kamodo-Core Remote Procedure Call (RPC) Update

- Improved code base released by Ensemble team via **Kamodo-Core Version 22.5.0.**
  - Adds RPC-based cloud & API Configurability,
  - Cross-language usage enabled (Fortran, C++ etc)
- This RPC specification, built on Cap'n Proto serialization protocol, provides a flexible and generic means for Kamodo end users to call containerized Kamodo functions
  - The KamodoClient, which subclasses Kamodo, behaves just like Kamodo core's base class, except that all interpolations and function evaluations will occur within the remote containers that implement them.

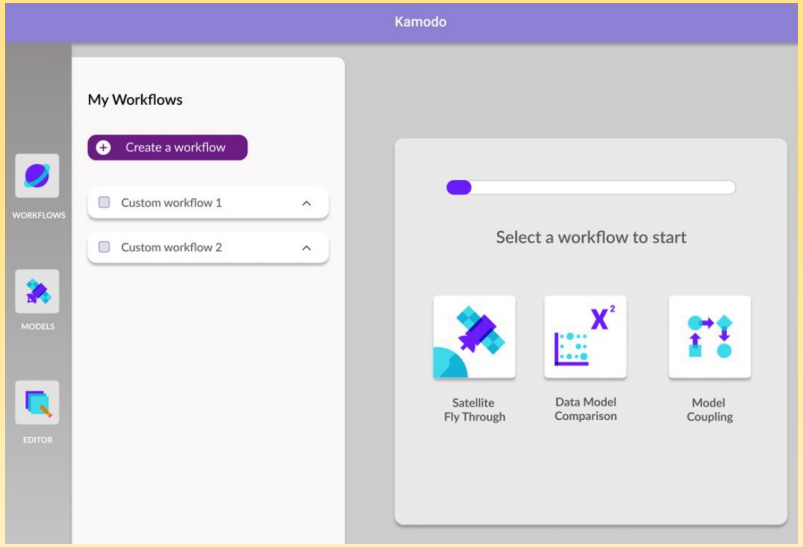


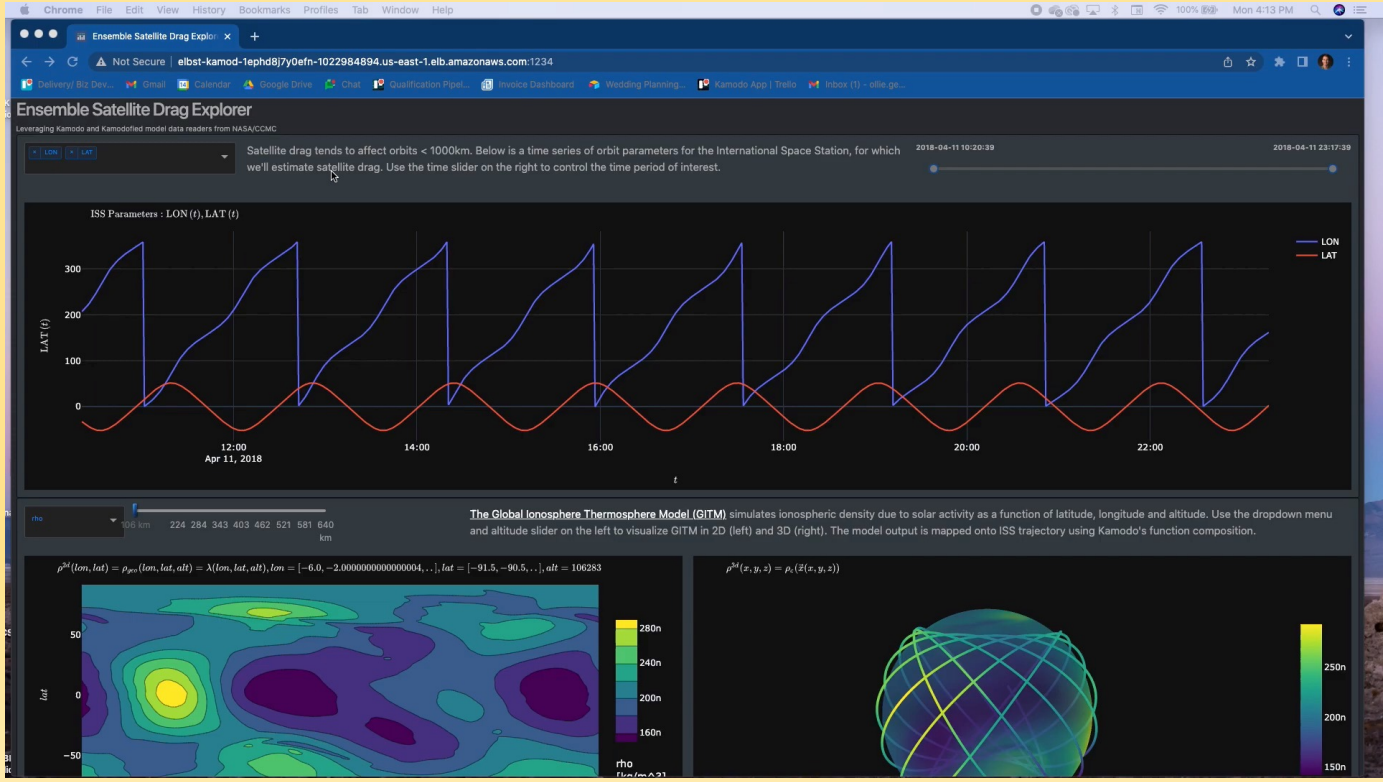
Binary Data Streaming: In addition to the basic python data types, functions can send and receive binary data.

This allows NumPy arrays and accompanying metadata to be communicated as quickly as possible without the need for JSON serialization on either end.

Kamodo-Core Code Free Interface

- Containerized custom workflows
- Interactive access to model data
- Great for commercial and education applications
- https://youtu.be/vpJgtAMCVvc





Kamodo: Future Plans

- **Open Science**: Improve documentation, build open access to compute capabilities next to model data, include miniature executable papers as tutorials.
- **New features**: Orbit propagators through pysat, heliospheric coordinate systems through SpacePy, image analysis through SunPy.
- CCMC **core improvements**: Expanded model library through collaborations, parallelized interpolators, HAPI layer on top of flythrough function, line-of-sight calculation tool.
- **Science research**: Constellation studies, expanded model-data comparisons, model-model interfaces for model coupling