Interfacing GEODYN and Kamodo

Z.C. Waldron[1], K. Garcia-Sage[2], E.K. Sutton[1], and J.P. Thayer[1]

[1]*CU Boulder Space Weather Technology, Research, and Education Center (SWxTREC)*

[2]*NASA Goddard, Space Weather Laboratory, Community Coordinated Modeling Center*

September 2021

Contents

1 Summary

This document goes through the details for how GEODYN has been interfaced with the CCMC's Kamodo— an atmospheric model API for Python. *The method described in this document is termed the Orbit Cloud Method and is used in lieu of a direct call to the Kamodo model so as to circumvent the complications of calling Python from FORTRAN.*

In summary, GEODYN has been interfaced to Kamodo through use of a method that we have termed the "Orbit Cloud Method". This method uses a pre-initialized run of GEODYN (using MSIS2.0) to get an `Init_Orbit` that serves as a best guess for what the orbit of the satellite will be for a given Kamodo Model. At each timestep of this orbit, we construct a cube of uncertainty around the location so that spatial variations when propagating the satellite will be captured within this cube. We refer to this orbit and its uncertainty cubes as an "Orbit Cloud". The orbit cloud's data points are fed into Kamodo from Python so as to make use of Kamodo's vectorization capabilities. In GEODYN's FORTRAN source code, a trilinear numerical interpolation is performed to extract a value from within the cube of uncertainty.

1.1 GEODYN Description

GEODYN is an orbit determination and geodetic parameter estimation program. Users of GEODYN input estimates of orbital parameters (such as the initial satellite state and solar radiation coefficients) and geodetic parameters (such as tracking station coordinates). GEODYN then computes orbits from the input parameters and can also compute theoretical values of satellite tracking observations using the input geodetic parameters. GEODYN can compare the theoretical values of tracking observations with real tracking observations to refine the input values of orbital and geodetic parameters.

Implementation In its current implementation on the CCMC AWS server, GEODYN is run through `Pygeodyn`, which wraps the FORTRAN code into an more easily accessible python program. Pygeodyn handles the necessary input files, setup scripts, and run sequences to execute a prepared run of GEODYN. It also handles the organization and selection of output files, and can be used to directly read desired datasets without the need to go through individual binary-to-ascii data dumps for the more pertinent GEODYN outputs. Wrapping the GEODYN software in a Python package opens the door for connections to programs such as Kamodo, which is also written in python.

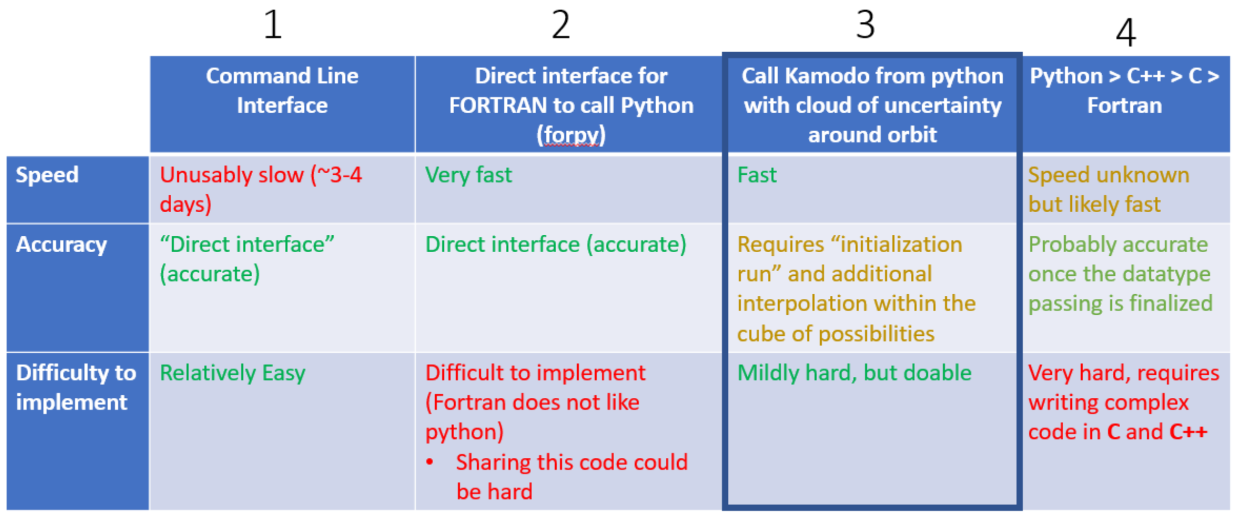
1.2 Kamodo Description

Kamodo is a new CCMC tool for access, interpolation, and visualization of space weather models and data in python. Kamodo allows model developers to represent simulation results as mathematical functions which may be manipulated directly by end users. Kamodo handles unit conversion transparently and supports interactive science discovery through Jupyter notebooks with minimal coding and is accessible through python. Kamodo is chosen for this project for its ability to offer model agnostic methods for reading data output from different model sources.

Implementation Kamodo is called using its Satellite Flythrough capabilities, in which a user is able to give some satellite ephemeris to Kamodo and Kamodo will return requested values for the chose model. We use this capability to request the Density at every timestep of the satellite's run.

2 Methodology

2.1 Other Interface Options



	1	2	3	4
	Command Line Interface	**Direct interface for FORTRAN to call Python (forpy)**	**Call Kamodo from python with cloud of uncertainty around orbit**	**Python > C++ > C > Fortran**
Speed	Unusably slow (~3-4 days)	Very fast	Fast	Speed unknown but likely fast
Accuracy	"Direct interface" (accurate)	Direct interface (accurate)	Requires "initialization run" and additional interpolation within the cube of possibilities	Probably accurate once the datatype passing is finalized
Difficulty to implement	Relatively Easy	Difficult to implement (Fortran does not like python) • Sharing this code could be hard	Mildly hard, but doable	Very hard, requires writing complex code in **C** and **C++**

The chosen method for connecting the GEODYN orbit propagation tool to Kamodo is to construct an orbit cloud of uncertainty around a pre-initialized orbit and then feed this orbit cloud (and its corresponding density values) into GEODYN. This method is chosen for its speed relative to calling the Kamodo API from the command line at every instance of the GEODYN orbit from FORTRAN. We found that each call to the command line (calling python from fortran) took approximately 1-2 seconds since all the python packages were re-imported at each call. This made the command line method unusable slow for connecting the two programs.

The Command Line Method The command line method is capable of being coded directly into the FORTRAN source code of GEODYN. From the `DRAG.f90` subroutine, one can identify which model is being requested and subsequently call the `KamodoModels_cl.f90` subroutine. From this subroutine, a string is constructed that sends a python command to the command line which contains the necessary inputs to the `Kamodo/SingleSatelliteFlythrough.py` function. The python-command is sent to the command line using the `EXECUTE_COMMAND_LINE()` function in FORTRAN. The python code then uses Kamodo to extract the sampled density at this single satellite locations/time from the chosen model and subsequently saves this data point to a text file before exiting python back to FORTRAN. The remaining lines in `KamodoModels_cl.f90` read the data point from the text file and performs any necessary unit conversions.

This method is accurate as it is a direct call to the model output, however it was determined to be unusably slow due to python needing to re-import all of its packages at each command line call from FORTRAN. Methods for forcing Python to remain open in the background were investigated, but eventually abandoned in favor of circumventing the FORTRAN-to-Python problem using the Orbit Cloud Method.

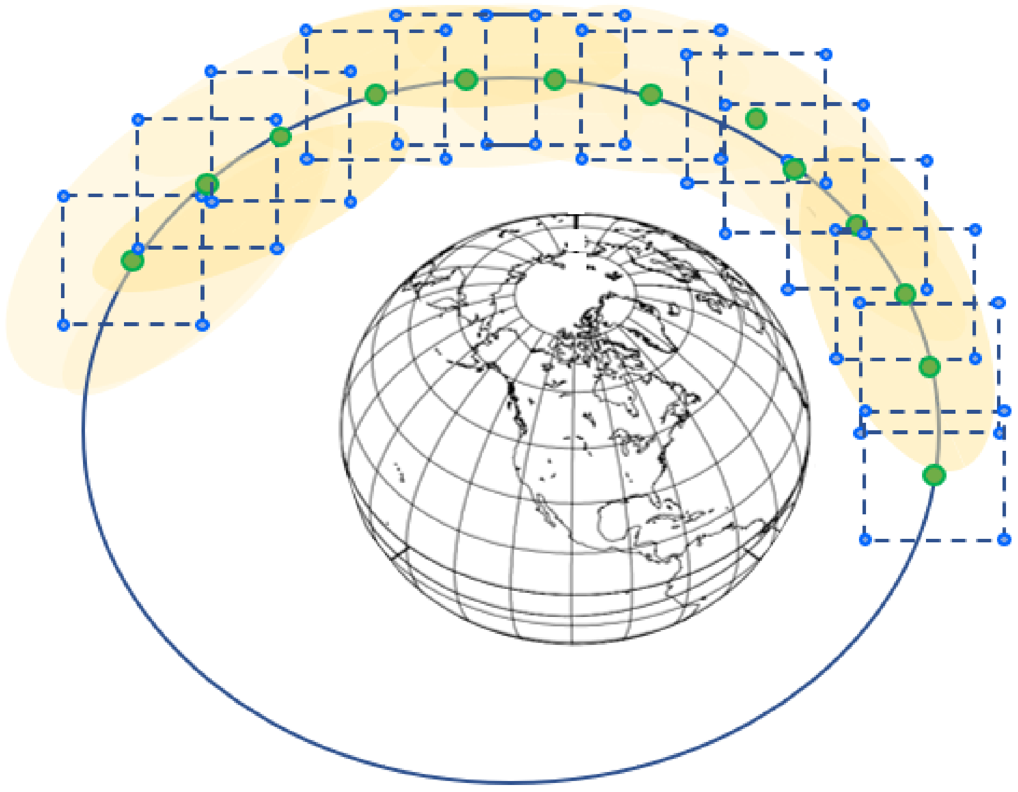
2.2 Orbit Cloud Method

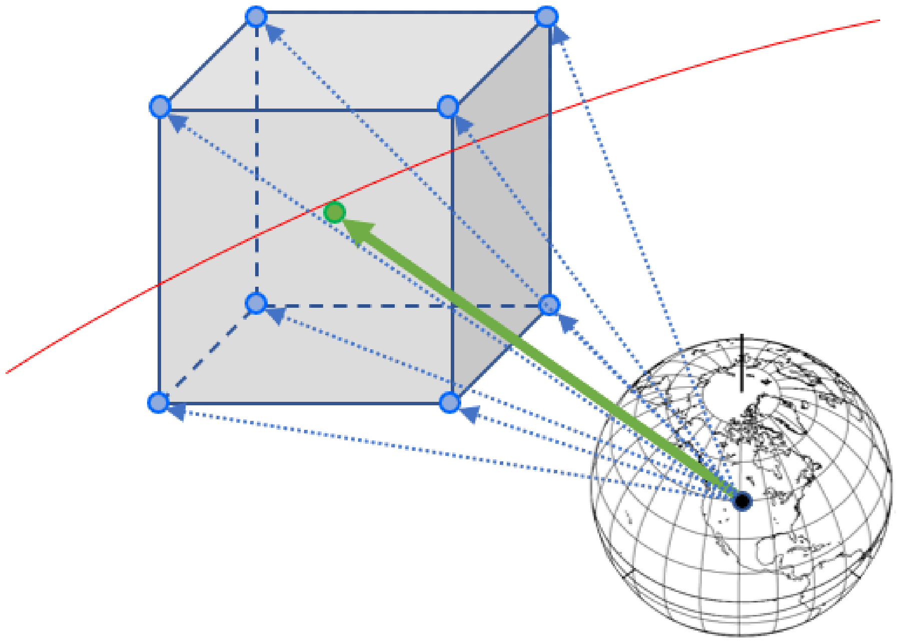
From Python

1. Pre-Initialize an orbit of the chosen satellite
   - Run GEODYN with MSIS2 to get a "best guess" for the orbit coordinates
2. Construct box of estimated orbits with uncertainties included
   - Using the "best guess" orbit, extend out the uncertainty of the coordinates to create a “cube of possible values” that TIEGCM might.
3. Plug Orbit Cloud into Kamodo
   - Plug the orbit + the uncertainty cube into Kamodo to extract the densities at all points.
   - From here, we have a cloud of possible density values from Kamodo at each timestep and within some standard deviation of possible values along the orbit of the spacecraft.

From FORTRAN

4. Run GEODYN with the TIEGCM option
   - Use a GEODYN friendly file to read the saved densities along the orbit into GEODYN as the program propagates.





Additional Notes and Features

- A feature can be added down the road that updates the cube size automatically if the cube is too small. For instance during a TIEGCM run, if the requested orbit location is outside of the constructed cube due to differences in the density models veering the orbit off course

- The uncertainty resolution (size of our cube) that we have chosen at this stage of testing (2 degrees for lon/lat and 1 kilometer for altitude) actually might be more than sufficient to capture variations in density. This is because the model resolution we are using for TIEGCM is 5°x5° so any variation in longitude or latitude will be captured to within that resolution. In the altitudinal domain, the scale height of the density is 50km at the orbiting altitude of ICESat2, so a resolution of 1km should be more than sufficient to capture its vertical variations. If we use the higher resolution TIEGCM outputs such as the double resolution (2.5°x2.5°) or higher we can scale the resolution to be more fine, accordingly.

3 Tests