The future of ARTS

Plans and progress for version 3

Richard Larsson

June 4, 2024

Overview

Work in progress

- What you are about to see is a work in progress.
- Some code snippets will not work in the future or simply have better solutions.
- This is about general ideas and concepts and why we have chosen to do this work.
- As such, some examples are numerically a bit silly.
- If you plan to continue using ARTS, you should pay attention as because the changes will be significant; some things will not work similar to the ARTS 2.6 way.
- We are having a look at all aspects of ARTS in this redesign.
- I hope that we can get some feedback on the direction of ARTS 3.

Operators

The problem today

- ARTS 2.6 has Agenda's to solve dynamic tasks. What is the the gravity at zero altitude? Call g0_agenda to compute g0!
- So what is the problem with Agendas?
  - Agendas are expensive to call.
  - Methods like z_fieldFromHSE take the g0_agenda to compute g0 and then adapted it for other positions.
  - Why not implement g_agenda or gm_agenda? Because it is expensive to call.

We are adding operators to provide cheap replacements of Agendas

- Operators are basically just passing $f(x, y, z, \cdots)$ into ARTS.
- These can be passed to ARTS method to handle tasks, similar to Agendas.
- They are basically ways to pass functions into ARTS to handle simple tasks.
- Simple examples include ways to compute the gravity field.
- Intermediate examples include computing the magnetic field using the builtin IGRF model.
- More complex examples includes using xarray to provide atmospheric field data via memory mapping the file.

Example

```
G, M = 6.673e-11, 5.972e+24

ws.gravity_operator = lambda z, la, lo: 9.8                           # 1
print(round(ws.gravity_operator(100e3, 0, 0), 3))

ws.gravity_operator = lambda z, la, lo: G * M / (6378e3 + z) ** 2    # 2
print(round(ws.gravity_operator(100e3, 0, 0), 3))

ws.gravity_operatorCentralMass(mass=M)                                # 3
print(round(ws.gravity_operator(100e3, 0, 0), 3))
print(round(ws.gravity_operator(100e3, 90, 0), 3))
```

Prints 9.8, 9.496, 9.498, and 9.561. 1) only returns a constant via the python lambda, 2) assumes spherical geometry via a python lambda, and 3) uses a builtin C++ method for the gravity field which is elliptical instead as shown by the printing.

Extending ARTS using the Operator mechanism

- External code can use ARTS operators.
- Alternatively, ARTS can use external code wrapped as operators.

Full Polarization, Full 3D, All the Time

The problem today

- ARTS 2.6 offers 1D-3D atmospheres in 1-4 Stokes parameters.
- That's 12(?!) different ways to perform core computations.
- Every single function must know how to deal with all those combinations.
- This is simply not maintainable for the future, we have to simplify the code base or no one will be able to maintain it and evolve it.

The spectral radiance is fully polarized

Transmission is from

$$\mathbf{T} = \exp\left(-\mathbf{K} r\right).$$

Beer's law for such a path is

$$\mathbf{I}_r = \mathbf{T}\mathbf{I}_0.$$

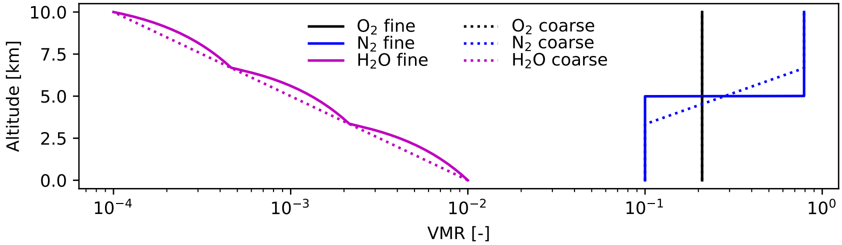
The spectral radiance is $\mathbf{I}$ is a 4-long Stokes vector. The transmission matrix $\mathbf{T}$ is a 4x4 matrix. The propagation matrix is a 4x4 matrix (though by memory layout, it contains only the 7 relevant variables).

The atmosphere as a 3D field operator

- The atmosphere is always a complete 3D field of *all* the properties it can contain.
  - User-defined top-of-the-atmosphere.
  - Point values at *any* altitude, latitude, and longitude coordinate below.
- Currently, the field handles the following properties:
  - Basic properties: temperature, pressure, wind, and magnetic fields.
  - Volume mixing ratios.
  - Isotopologue ratios.
  - Non-local thermodynamic equilibrium scaling.
  - Particulate properties.
- Each data property (e.g., the temperature field) can be stored as:
  - A 3D gridded field. As in ARTS 2.6 for e.g., temperature.
  - A constant numeric. As in ARTS 2.6 for e.g., isotopologue ratios.
  - A ternary operator. Not available in ARTS 2.6.
- Interpolation and extrapolation are handled per data property.

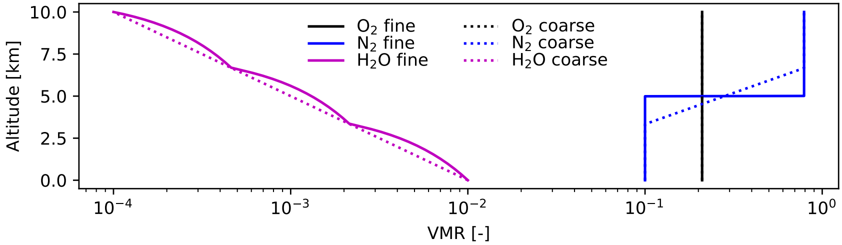
Example - one of each data type

```
data = np.logspace(-2, -4, 4).reshape(4, 1, 1)
gn = ["Altitude", "Latitude", "Longitude"]
g = [np.linspace(0, 10e3, 4), [0], [0]]
ws.atmospheric_field[H2O] = GriddedField3("VMR", data, gn, g)
ws.atmospheric_field[O2] = 0.21
ws.atmospheric_field[N2] = lambda z, la, lo: 0.1 if z < 5e3 else 0.79
```



Example - using xarray and potentially memory mapping

```
GriddedField3("VMR", data, gn, g).to_xarray().to_netcdf("testfile.nc")
def interp(alt: float, lat: float, lon: float):
    da = xr.load_dataarray("testfile.nc")
    return da.interp({"Altitude": alt}, assume_sorted=True).data[0, 0]
ws.atmospheric_field[H2O] = interp
```



Using builtin methods

- We will have choices of builtin methods to compute the atmospheric properties functionally as well.
- In today's code, we have:
  - A method to get the magnetic field using the IGRF model at the sample pont.
  - A method to compute the pressure field based on atmospheric composition and temperature using hydrostatic or hypsometric approximations.
- We are more than happy to add more builtin methods and helper functions as pure python if provided.

No Main Grid

The problem today

- There are too many variables in ARTS 2.6.
- We have pressure, latitude, longitude, frequency, azimuth (in and out), zenith (in and out), species, and...
  - Want to perform the most simple spectral radiance calculations in ARTS 2.6? Use iyEmissionStandard. It takes 35 input arguments and gives 13 output arguments.
  - Want to perform the most simple spectral radiance calculations in ARTS 3? Use spectral_radianceClearskyEmission. It takes 11 input arguments and gives 2 outputs.
- These variables depend on eachother, but not always in a straightforward manner.
- Take the magnetic field and the wind field. They are both fields of 3D vectors, but input as a total of 6 arguments. These arguments must match the pressure, latitude, and longitude grids. Meaning we need 9 arguments to even use and understand them properly. In ARTS 3, we make them part of the atmospheric field.

There is no main grid

- The grids are part of the data.
  - The atmospheric field.
  - The surface field.
  - The sensor properties.
  - ...
- API consequences:
  - The atmosphere and the surface fields have their own (alt-,) lat-, and lon-grids.
  - The sensor elements have their own frequency grid.
- Retrieval consequences:
  - Atmospheric and surface parameters are retrived on the field grid.
  - Sensor parameters are retrieved on... what grid? This is still an open design question.

Current Status

What is working

- The python interface. Much more so than in ARTS 2.6. (No more ".value"!)
- Full emission and transmission in a clearsky atmosphere. Including a pure first-order Rayleigh-scattered sun.
- The atmospheric field and surface fields are available.
- An experimental implementation of the sensor design.

What is not working

- The OEM system.
- The scattering calculations.
- The active sensors.
- Most path calculations.
- The non-LTE solver.

Questions?