ReFRACtor

Reusable FRamework for Atmospheric Composition

James McDuffie

ROSES 16-AIST-0019

INTRODUCTION
ReFRACtor

- Reusable FRamework for Atmospheric Composition
- Fork and descendant of operational OCO-2 retrieval software
- Descendant of earlier code bases and over 10 years of work
OBJECTIVE

Develop an extensible multi-instrument atmospheric composition radiative transfer (RT) and retrieval framework to reduce the cost and risk of Level-2 development for future atmospheric Earth science missions.
## TEAM MEMBERS

<table>
<thead>
<tr>
<th>Team Member</th>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>James McDuffie</td>
<td>PI / Software engineering</td>
</tr>
<tr>
<td>Kevin Bowman</td>
<td>Co-I / Science</td>
</tr>
<tr>
<td>Jonathan Hobbs</td>
<td>Co-I / Uncertainty quantification</td>
</tr>
<tr>
<td>Vijay Natraj</td>
<td>Co-I / Radiative transfer</td>
</tr>
<tr>
<td>Edwin Sarkissian</td>
<td>Retrieval implementation</td>
</tr>
<tr>
<td>Matthew Thill</td>
<td>Signal processing</td>
</tr>
<tr>
<td>Sebastian Val</td>
<td>Software engineering</td>
</tr>
</tbody>
</table>
GOALS

- Serve as a common, reusable and extensible framework for processing atmospheric composition Level-1 data through to Level-2
- Support the data fusion of observations from multiple instruments in joint retrievals into multi-species data products
REUSABILITY

• New instrument missions can start with a framework that has tested and verified algorithms
• New missions can focus on the novel algorithmic pieces unique to their instrument
• Eliminates duplication, and lowers costs while providing critical capabilities that support missions and longer term national needs
REUSABILITY

• Reusable and extensible allowing different instrument teams to use the same code base
• Multi-instrument not only in the sense that different instruments can use the same software, but also in the sense that data products can be enhanced from multiple instruments through joint retrievals
DUAL ROLES

- Use in operational processing of remote sensing measurements
- Use of framework components piecemeal for research
DATA FUSION

- Support data fusion of radiance measurements from multiple instruments through joint retrievals
- Data fusion of multiple instruments provides products with greater vertical sensitivity
- Joint retrievals additionally reduce measurement uncertainty.
- Increasing the density of observations with complementary time and spatial scales can improve our understanding of dynamic Earth systems such as the carbon cycle.
FAR FUTURE

This framework could enable an atmospheric science analytic center to support a common code base for joint retrievals and data fusion among multiple instrument teams.
SCIENCE OBJECTIVE

Determine atmospheric composition constituents for a column of air observed by a satellite, aircraft or uplooking instrument.
LIGHT PATHS

Many paths for radiation to enter instrument: thermal, solar absorption and scattering. Contributions from atmosphere and surface.
ATMOSPHERIC REMOTE SENSING

- Determine the concentration of gases in a vertical column of the atmosphere by the measurement of solar radiation and emission of thermal energy
- Takes advantage of the differences in the interaction of light and molecules at different wavelengths and pressures
Using radiative transfer mathematics we can model the propagation of radiation through the atmosphere to derive a synthetic spectra.
Using optimal estimation, update a state vector primed by our best initial guess of the atmospheric components to get the smallest possible difference between modeled and measured spectra.
COMPONENTS
DATA FLOW

Initial guess, Ancillary data

Atmospheric State

Forward Model

Modeled Radiances

A priori, Constraints

Optimal Estimation

Measured Radiances

Error Analysis

Output Product
FORWARD MODEL

Models the spectra an instrument would have measured given a particular combination of atmospheric contents and viewing geometry using a physics based software.
The largest important components in the forward model are a radiative transfer model and instrument model.
The radiative transfer component creates a normalized monochromatic radiance that accounts for thermal emissions, molecular absorption and particle scattering of surface and solar radiation.
Applies the effects of the modifications to radiation as a result of the instrument performing the measurement.
Radiative transfer radiances are a model of the radiation before it enters the instrument.

Instrument model radiances represent the radiation after going through the instrument optical path to the detector.
• Repeatedly evaluates the forward model given some constraints and a priori information
• Compares modeled and measured radiances to make changes to the atmospheric state for subsequent iterations.
• Exits when the cost function meets a defined convergence criteria
RETRIEVAL APPROACH

• Use optimal estimation (maximum a posteriori) approach described by Rodgers in *Inverse methods for atmospheric sounding* [2000]

• Various interchangeable Non-Linear Least Squares Solvers (NLLS) available in framework
DESIGN PRINCIPLES

- Use abstract interfaces to decouple behavior from implementation
  - Open/closed principle
  - Dependency inversion
- Use Adapter pattern to reuse existing code
OPEN/CLOSED PRINCIPLE

- “Open for extension”
  Behavior can be extended as requirements change
- “Closed for modification”
  Working, tested software should not change to meet requirements
OPEN/CLOSED PRINCIPLE

• Core ideas at the heart of good object oriented design
• Use abstract base classes to define an interface
• Interfaces define possible behaviors of implementations
New implementations only require adding new classes that conform to a predefined abstract interface.
DEPENDENCY INVERSION

- High level modules should not depend on low level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Instead, details should depend on abstractions.
Place interfaces between components to eliminate dependence on a specific implementation.
LANGUAGE CHOICE

• We chose C++ for the top level
  ▪ Can express our object oriented design with it
  ▪ Easy to interface with other languages
  ▪ Well established with mature third party libraries

• Kept major portions, such as RT in Fortran
  ▪ Use Fortran 2003's iso_c_bindings
  ▪ Built interfaces and drivers to conform to our design
  ▪ No loss in speed due to memory copying
THIRD PARTY SOFTWARE

- Blitz++
- Eigen
- Boost C++ Libraries
- GNU Scientific Library
- HDF5
- Python
PYTHON INTERFACE

python™
REASON

- Framework is complex collection of modules
- Python flexibility
  - Try out new algorithms and approaches
  - Speeds up prototyping of algorithmic additions
  - Allows using components piecemeal
IMPLEMENTATION

- Wraps C++ classes using SWIG
- Python classes can be made to stand in for C++ classes
- Doxygen documentation is available through standard Python help system
CONFIG FACTORY LAYER

• Provides a "business" logic layer connecting wrapped C++ classes with inputs/outputs
• Structured as a logical hierarchy that matches the framework interface design
• Simplifies connecting components together and ensuring their input types through "Creator" classes
• Provides a keyword/value interface to connecting data and configuration into the framework
• Does not tie the hands of users by enforcing a strict expectation of inputs, ie very flexible
Each oval evaluates to data or a class
Highlights the different pieces of information needed to construct an example atmosphere
EXAMPLE CONFIG PIECE

'absorber': {
    'creator': creator.absorber.AbsorberAbsco,
    'gases': ['CH4', 'CO', 'CO2', 'O3', 'H2O', 'HNO3', 'NO', 'NO2', 'N2O', 'NH3'],
    # Default gas definition when a custom block named for the gas is not supplied
    'default_gas_definition': {
        'creator': creator.absorber.AbsorberGasDefinition,
        'vmr': {
            'creator': creator.absorber.AbsorberVmrLevel,
            'value': {
                'creator': creator.absorber.GasVmrApriori,
            },
        },
        'absorption': {
            'creator': creator.absorber.AbscoHdf,
            'table_scale': 1.0,
            'filename': '{absco_base_path}/{gas!!}.cris.h5'
        }
    }
}
from refractor.factory import process_config
import cris_config
config_inst = process_config(cris_config.config_def)
```python
>>> pprint(config_inst, indent=2)

{ 'atmosphere': <refractor_swig.atmosphere_oco.atmosphereoco; proxy="" of="" <swig>"" object>,
  'common': { 'absco_base_path': '/mnt/data1/absco/cris',
    'band_reference': <refractor_swig.array_with_unit.arraywithunit_double_1; proxy="" of="" <swig>"" object>,
    'constants': <refractor_swig.default_constant.defaultconstant; proxy="" of="" <swig>"" object>,
    'desc_band_name': ['LW', 'MW', 'SW'],
    'hdf_band_name': ['LW', 'MW', 'SW'],
    'num_channels': 3,
    'stokes_coefficients': array([[1., 0., 0., 0.],
                                 [1., 0., 0., 0.]]),
  }
  'forward_model': <refractor_swig.standard_forward_model.standardforwardmodel; proxy="" of="" <swig>"" object>,
  'instrument': <refractor_swig.ils_instrument.ilsinstrument; proxy="" of="" <swig>"" object>,
  'radiative_transfer': <refractor_swig.lidort_rt.lidortrt; proxy="" of="" <swig>"" object>,
  'scenario': { 'altitude': <refractor_swig.array_with_unit.arraywithunit_double_1; proxy="" of="" <swig>"" object>
  }}
```
from refractor import framework as rf
from matplotlib.pyplot import *

fm = config_inst.forward_model
meas_grid = fm.spectral_grid.low_resolution_grid(channel_idx).data

channel_index = 0
rad = fm.radiance(channel_idx)

plot(meas_grid, rad.spectral_range.data / 1e-3)
ylabel("$ mW / sr / m^2 / cm^{-1} $")
xlabel("Wavenumber (cm$^{-1}$)")
RESULT
AUTO DERIVATIVES

\[ f(x) \{ \ldots \}; \]

\[ y = f(x) \]

\[ df(x) \{ \ldots \}; \]

\[ y' = f'(x) \]

Automatic differentiation

human programmer

symbolic differentiation
(human/computer)
TECHNIQUE

• Use a technique called Rall numbers, as described in Scientific and Engineering C++ by Barton and Hackman
• Works well with code that calculates Jacobians in other ways
• Only adds a bit more arithmetic operations than original equations, although is not as fast as hand-coded derivative calculations
• For locations in the code that profiling show as bottlenecks, we can replace the auto derivate calculation with hand-created code that gives the same results
• Easy to create derivatives for equations in code that does not already have it, just by changing types
IMPLEMENTATION

• Instead of using a standard C double we use a new C++ class AutoDerivative that keeps track of both a value and the gradient of the value with respect to the state vector elements.

• All normal mathematical functions are overloaded to use the chain rule to propagate the gradient through the calculation.

• Instead of using normal Blitz++ array class use a ArrayAD class to keep jacobians with matrix values.
CLASS OVERVIEW

<table>
<thead>
<tr>
<th>FullPhysics::AutoDerivative&lt; T &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ AutoDerivative()</td>
</tr>
<tr>
<td>+ value()</td>
</tr>
<tr>
<td>+ gradient()</td>
</tr>
<tr>
<td>+ is_constant()</td>
</tr>
<tr>
<td>+ number_variable()</td>
</tr>
<tr>
<td>+ operator+=()</td>
</tr>
<tr>
<td>+ operator*=</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FullPhysics::ArrayAd &lt; T, D &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ ArrayAd()</td>
</tr>
<tr>
<td>+ cols()</td>
</tr>
<tr>
<td>+ depth()</td>
</tr>
<tr>
<td>+ is_constant()</td>
</tr>
<tr>
<td>+ value()</td>
</tr>
<tr>
<td>+ jacobian()</td>
</tr>
<tr>
<td>+ number_variable()</td>
</tr>
<tr>
<td>+ operator(int1)</td>
</tr>
<tr>
<td>+ operator(int1, int2)</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
INTEGRATION WITH OTHER TECHNIQUES

- Our radiative transverse code, LIDORT is already calculating Jacobians.
- We calculate the Jacobian of the parameter up to the point where LIDORT is called.
- Use LIDORT itself to propagate through the RT calculation.
- Take the results of LIDORT, wrap them up as AutoDerivative objects, and continue using this to propagate through the remaining forward model calculation.
Example

double x = 1;
double y = 2;
double z = x * x + x * y + 2 * y * y;
// Result is 11

AutoDerivative<double> x(1, 0, 2);
AutoDerivative<double> y(2, 1, 2);
AutoDerivative<double> z = x * x + x * y + 2 * y * y;
// Result is 11 gradient is [4, 9]
MULTIPLE INSTRUMENT SUPPORT

Earth Science Missions

ISS Instruments
CATS, LIS, SAGE III
TSIS-1, OCO-3, ECOSTRESS, GEDI
CLARREO-PF

JPSS-2 Instruments
RBI, OMPS-Limb

Formulation
Implementation
Primary Ops
Extended Ops
INSTRUMENTS

ReFRACtor Framework

CrIS  OMPS  OCO-2  GOSAT  CARBO  GeoCARB

CrIS + OMPS

In Progress
Planned near term
Planned longer term
Heritage
IMPLEMENTATION

- Adding support for multiple instruments through framework’s abstract interfaces and separate source code repositories
- Code is added instead of modifying framework
- Each instrument specific repository just has the code specific to that instrument plus a configuration
NEEDED FOR EACH INSTRUMENT

• Measured data reader, ie Level 1B reader
• Instrument model
  ▪ Instrument Line Shape
  ▪ Instrument radiance effects
• Domain specific radiative transfer algorithms
• Absorption Coefficient tables
CURRENT WORK
FRAMEWORK DEVELOPMENT

- Took heritage repository from OCO-2, cleaned up and moved out non reusable components
- Changed the build/compilation system to use CMake, a more modern and extensible compilation system
- Created Python configuration layer for building “business logic” of instrument implementations
INSTRUMENT IMPLEMENTATIONS

- Cross-track Infrared Sounder (CrIS)
  - Built Level 1B data reader
  - Built configuration
  - Working on adopting instrument team’s instrument line shape (ILS) data
- Carbon Balance Observatory (CARBO)
  - Created prototype Python only configuration for use in instrument simulations
- Orbiting Carbon Observatory (OCO-2)
  - Inherited, but needs to be put in place in a separate repository
RADIATIVE TRANSFER

- Upgraded interfaced versions of existing radiative transfer software to latest versions
  - LIDORT
  - Two stream model
- Integrated support for performing thermal infrared radiative transfer calculations
  - OCO-2 code has only been using in the near infrared
RETRIEVAL METHODS

- Upgraded existing Non-Linear Least Squared (NLLS) solvers by integrating newer versions with several more years of work on them beyond OCO-2 versions
- Added additional NLLS solvers that make different assumptions about the problem state
- Working on integrating white-box implementation of Moire's algorithm
UNCERTAINTY QUANTIFICATION

- Created a generalized Python code for simulation state vectors from mixture of statistical distributions
- Creating a report on canonical examples for doing Uncertainty Quantification, focused on implications for algorithm design
  - How is bias/variance impacted when selected state vector elements are fixed and not retrieved?
  - What is the impact of fast/simplified RT in the retrieval when true forward model is more complex?
- Investigating simplified RT usage of state vector mixtures
NEAR TERM ACTIVITIES

- Software for computing absorption coefficient (ABSCO) tables from infrared to ultraviolet spectral regions
- Integration of a radiative transfer speed up method using principal component analysis
- OMPS instrument implementation
- Improve retrieval strategy techniques
END
TESTING
UNIT TESTING

- Using the Boost Unit Test Framework
- Tests individual pieces of the system, for example individual classes
- Unit tests are good at:
  - Correct handling of error conditions
    Ex: Out of bounds, bad values
  - Detecting memory leaks in component
  - Repeatability of calculation
    Do we get the same answer today as last week?
  - Comparison between calculations
    Ex: Jacobian calculation vs. finite difference Jacobian
Jacobian

- Unit tests are bad at anything that involves analysis.
  We are happy to get the same wrong answer every time.
#include "auto_derivative.h"
#include "unit_test_support.h"

using namespace FullPhysics;
using namespace blitz;

BOOST_FIXTURE_TEST_SUITE(auto_derivative, GlobalFixture)

BOOST_AUTO_TEST_CASE(basic_test)
{
    double xv = 2; double yv = 3;
    AutoDerivative<double> x(xv, 0, 2);
    AutoDerivative<double> y(yv, 1, 2);
    AutoDerivative<double> z;

    z = 2 * x + y;
}
UNIT TEST EXAMPLE

$ make fast_check run_test=auto_derivative/basic_test
...

0% 10 20 30 40 50 60 70 80 90 100%
|----|----|----|----|----|----|----|----|----|----|
Running 1 test case...
*********************************************************************************

*** No errors detected
SUPPORT TOOLS
INSTRUMENT DETAILS
CROSS-TRACK INFRARED SOUNDER (CRIS)

- Fourier transform spectrometer
- LWIR (9.14 - 15.38 μm)
- MWIR (5.71 - 8.26 μm)
- SWIR (3.92 - 4.64 μm)
- Launched October 28, 2011 on S-NPP. Also 2017 on JPSS-1
OZONE MAPPING PROFILER SUITE (OMPS)

- Six-slit prism spectrometer.
- Single, two-dimensional (740×340) charged-coupled device (CCD) focal-plane array
- Spectral range from 290 to 1000 nm
- Launched October 28, 2011 on S-NPP. Also 2017 on JPSS-1
HISTORY
HISTORY

- Changes for GOSAT processing begins May 2009
- Mike Smyth joins team November 2009
- Start using Git, Feb 2010
- Last Fortran only version 2.06.02 March 2010
- Refactoring started March 2010
- First refactored version 2.07.00 July 2010
- Lua configuration introduced June 2011
- OCO-2 algorithmic work begins Dec 2011
- OCO-2 Launched on July 2, 2014
- ReFRACtor Project begins September 14, 2017
REFERENCES


