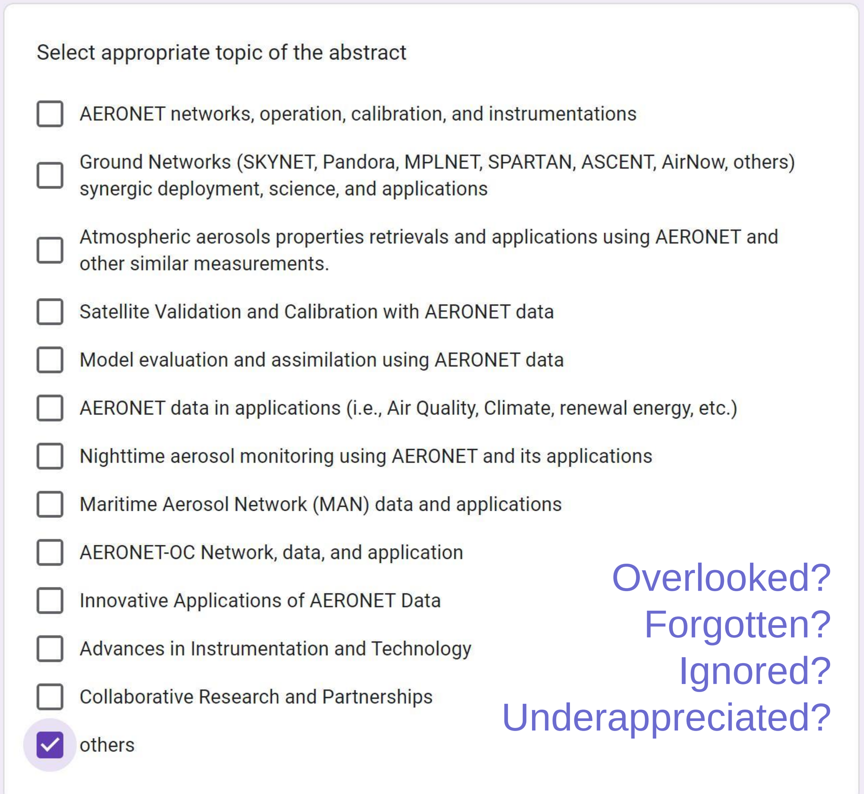
AERONET Project: The Next 30 Years of [Scientific] Software Development

Sergey Korkin (UMBC & NASA GSFC, sergey.v.korkin@nasa.gov), **Alexei Lyapustin** (NASA GSFC), and **Brent Holben** (NASA GSFC)

Abstract & Motivation

Rigorous protocols for instrument absolute calibration and data processing, developed and maintained by a dedicated international team for over three decades, have established AERONET as the global standard for aerosol research. ***Software has been, and will continue to be, a crucial contributor to the project's success***.

How about [scientific] software ?



Overlooked?
Forgotten?
Ignored?
Underappreciated?

> *"Scientists spend an increasing amount of time building and using software. However, most scientists are never taught how to do this efficiently. As a result, many are unaware of tools and practices that would allow them to write more reliable and maintainable code with less effort.*
> *…*
> ***Write program for people, not computers.*** *… "*
>
> Wilson + 12 (Canada, UK, US), "**Best Practices for Scientific Computing**", PLOS Biology (2014)

Maybe "it works" well and needs no further development ?

> *"The more frequently a program is changed, the more difficult it is to maintain its correctness. Scientific programs are frequently changed throughout their lifetimes, not just when they're young. For most scientific programs, the rate of change doesn't decrease significantly even after many years.* ***Like sharks, scientific programs that aren't moving are dead.****"*
>
> Dubois, "**Maintaining correctness in scientific programs**", Computing in Science & Engineering (2005)

AERONET relies on various software including data processing (primarily implemented in C-language), and research (forward and inverse modelling - mostly in Fortran). This disparity between C and Fortran, and the lack of support for modern software development tools (Python, Intel MKL high-performance mathematical library, Doxygen) disrupts seamless data flow and hampers research progress.

Furthermore, scientific software serves both a research tool and an object of study. As science evolves, the software undergoes modifications, often incorporating "temporary" patches that persist indefinitely within the codebase without proper documentation. Decades of extensive development without systematic cleanup inevitably turn software into "black boxes" with "spaghetti code".

> *"…several authors have argued that the "gap" or "chasm" between software engineering and scientific programming is a* ***serious risk*** *to the production of reliable scientific results, as demonstrated in a number of case studies."*
>
> Storer, "**Bridging the Chasm: A Survey of Software Engineering Practice in Scientific Programming**", ACM Computing Surveys (2017)

Our presentation outlines updates to scientific software, relevant to AERONET, over the past few years, including radiative transfer solver, atmospheric absorption spectroscopy, and light scattering by spheroids. ***We will share ideas based on our experience to stimulate discussion on enhancing and advancing one of the AERONET project's critical components***.

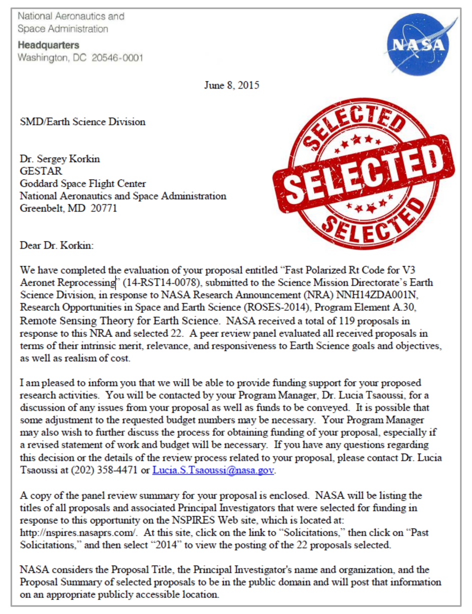
> *"A* ***critical challenge*** *in scientific computing is balancing developing high-quality software with the need for immediate scientific progress."*
>
> Adorf et al., "**How to Professionally Develop Reusable Scientific Software—And When Not To**", Computing in Science & Engineering, (2019)

> *"…we also demonstrate how moving to programming languages with high momentum, like* ***modern C++****, can help improve the sustainability, interoperability, and performance of research software."*
>
> Anzt et al., "**The Journey: Improving Software Portability, Productivity, and 100 x Performance**", Computing in Science & Engineering (2024)

Roberts "**The publication of scientific Fortran programs**" (1969) & "**Practical techniques in computer programming**" (1971) in Computer Physics Communications

Radiative Transfer (RT): Multiple Scattering of Sunlight

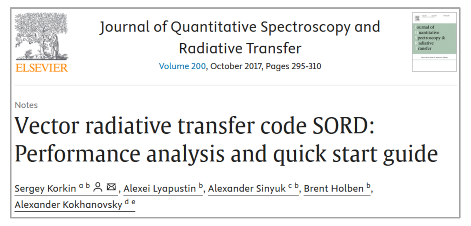
2015 → **2017** → **2020** → **2024**



ROSES RST Proposal:

"*Fast Polarized RT Code for V3 AERONET Reprocessing*"

PI: S. Korkin, Collaborators: A. Lyapustin, A. Siniuk, and B. Holben



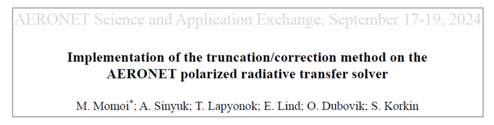
vRT solver SORD:

- Successive orders (replaced the original 🇫🇷 SOS vRT)
- Fortran 90/95
- Emphasis on the code **"quality"** for easy support:
  - Readability, comments
  - Structure
  - 50+ Tests (automatic mode)
  - " *Other people will read it* "
- Interface between the solver and inversion code
- Publicly available: https://github.com/korkins/SORD_JQSRT_2017

SORD supported **AERONET V3 reprocessing**

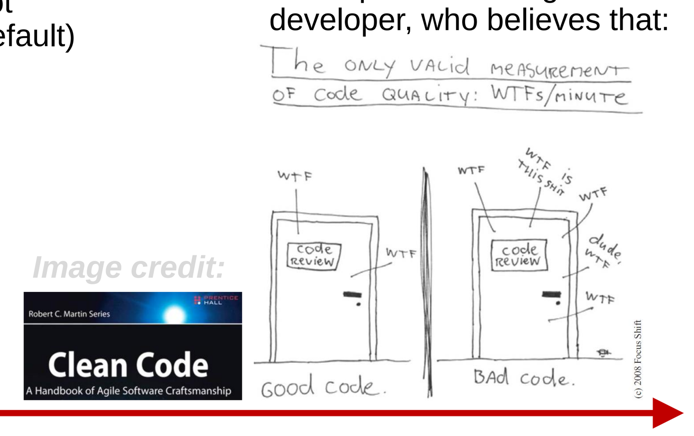
"We want it faster " (never ending requirement) & we need:

- to run it on a supercomputer
- to add fluxes, mix land & ocean
- TOA reflectance (was not provided on output by default)
- output "in the middle" of atmosphere (MLO observatory), etc.
- Multiple SZAs for MLO calibration (averaging)
- Add Cirrus to Aerosol & Rayleigh mix



Faster SORD:

- Did you see a poster by M. Momoi (GRASP) on Tuesday?
- Extension of Nakajima & Tanaka's technique (1988); with polarization Ota et al (2010) – both utilize DOM
- Phase matrix truncation + correction of higher orders of scattering (3 orders)
- No help from the original developer, who believes that:

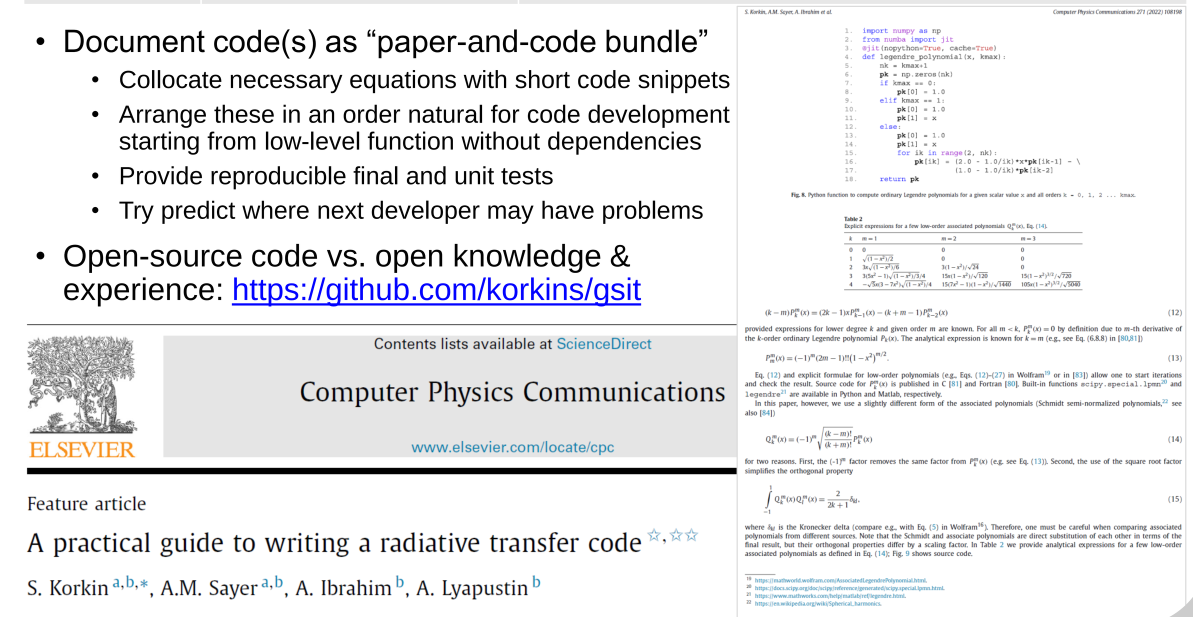


Mid-2018: end of the proposal = end of support for RT code. If HQ supports AERONET shouldn't the code get something?

Proposed To-do List for AERONET's RT Code (Solver):

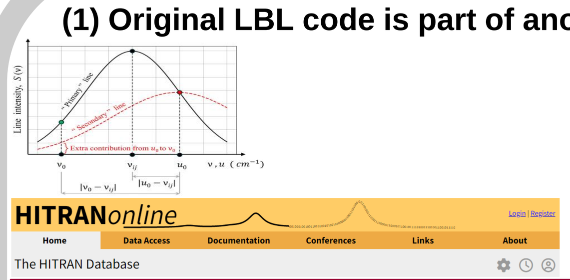
Job Type	Task	Purpose
Software	Translate into C	Seamless integration with data processing part
Software	Documentation	Future support & development cost minimization
Software	GitHub / GitLab	Same + debugging + NASA's Open-Source Science
Software	Professional code review	Bring expertise from a skilled software engineer
Model	Spherical-shell	Low SZA & VZA; hybrid scan
Model	Linearization	Jacobians for inversion
Model	Output at given height	Elevated stations (e.g., MLO)
Science	Skip polarization in higher scattering orders	Further acceleration
Computing	Interface for parallel runs	Faster calculation of the Jacobians and/or data processing
Computing	Support of GPU	Yet more faster runs

- Document code(s) as "paper-and-code bundle"
  - Collocate necessary equations with short code snippets
  - Arrange these in an order natural for code development starting from low-level function without dependencies
  - Provide reproducible final and unit tests
  - Try predict where next developer may have problems
- Open-source code vs. open knowledge & experience: https://github.com/korkins/gsit



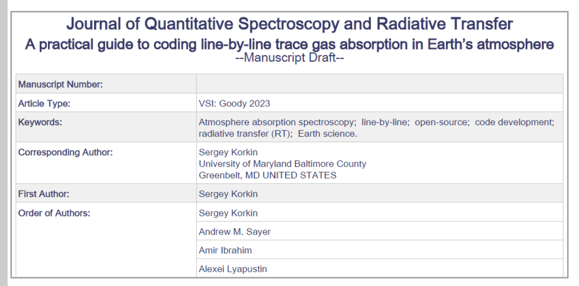
LBL Atmospheric Absorption

(1) Original LBL code is part of another package:

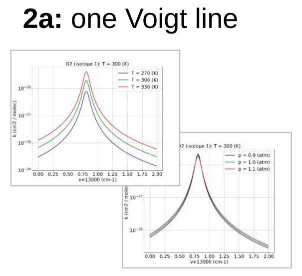



```
7113000.816219 2.708E-27 1.740E-02.04580.047 1814.01040.67-.007400        b       1       X       1               R 13Q 14      d54644449 5 5 3 1 1     29.0    29.0
```

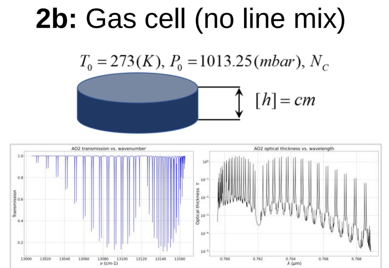
(2) Paper under review: "*A Practical Guide to Coding Line-by-line Trace Gas Absorption in Earth's Atmosphere*"



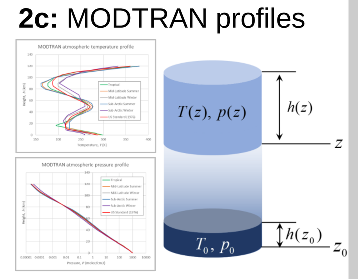
2a: one Voigt line



2b: Gas cell (no line mix)

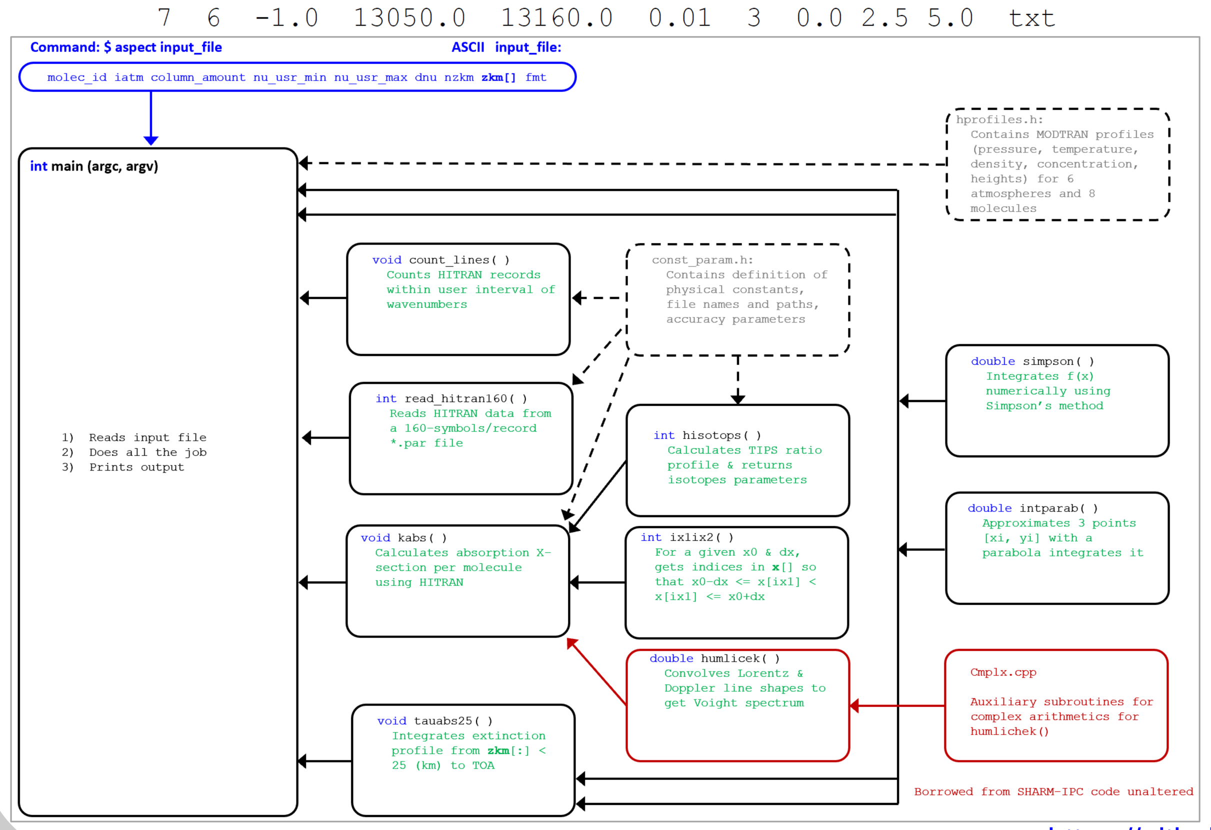


2c: MODTRAN profiles

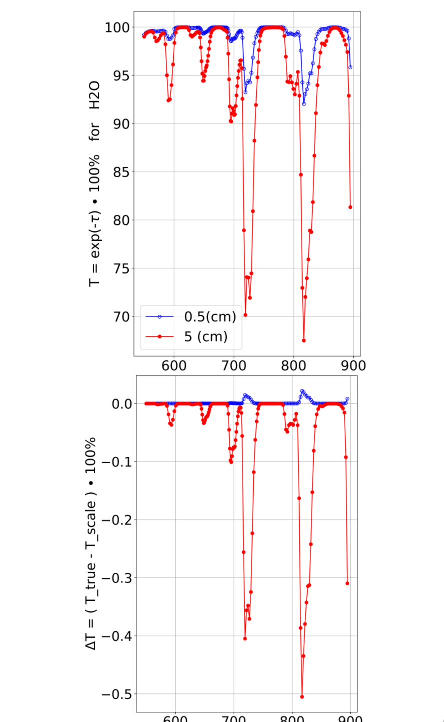


(3) Open-source code for LBL absorption in gas cell (`GCELL.cpp`) and atmosphere (`ASPECT.cpp`):

```
molec_id iatm column_amount nu_usr_min nu_usr_max dnu nzkm zkm[] fmt
       7    6          -1.0    13050.0    13160.0 0.01    3 0.0 2.5 5.0  txt
```



Practical example for OCI: scaling of OT(WV) = WV x OT(1.43 cm) / 1.43 vs. true calculation

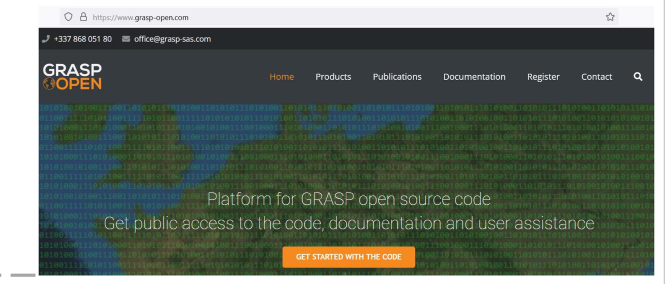


https://github.com/korkins/aspect_gcell

Light Scattering by Spheroids

Acknowledgments:

- SK received FORTRAN sources from **T. Lapyonok & O. Dubovik** in September 2012 (**AL** recived it way before)
- **MAIAC** has used the sources since then: **SK** – as is, **AL** – dropped polarization and converted the sources into C
- From **O. Dubovik** (private communication 2023/10/23 over email): "*…we do have new version and plan to have radically new one …*"
- Anyone interested should refer to the **GRASP** website for a newer version of the (so called) spheroidal or **DLS** package

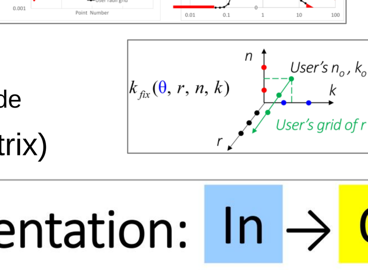


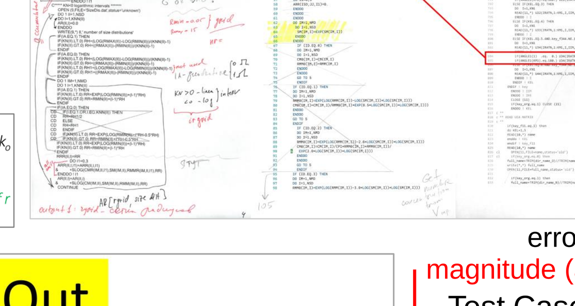
Reasons for refactoring:

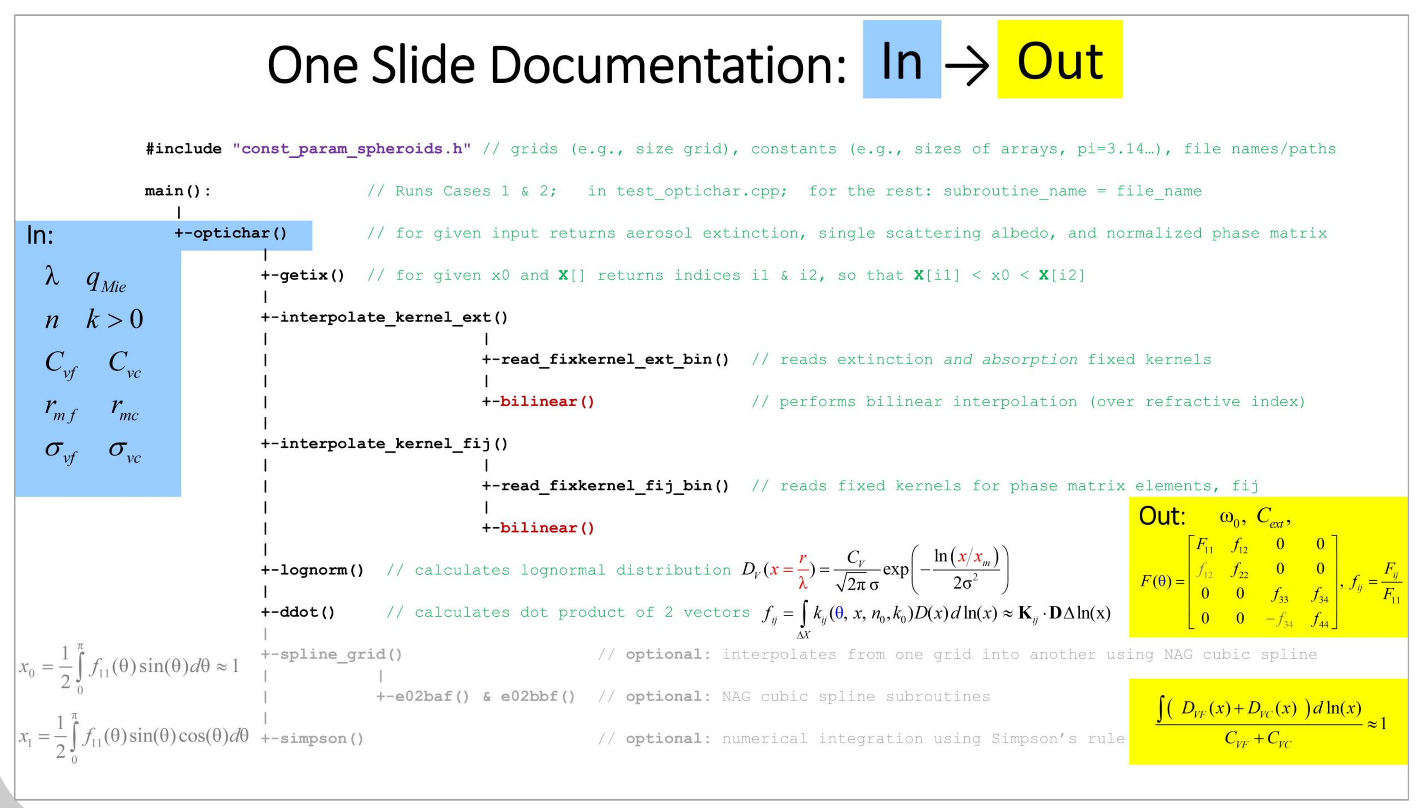
- SK is learning things

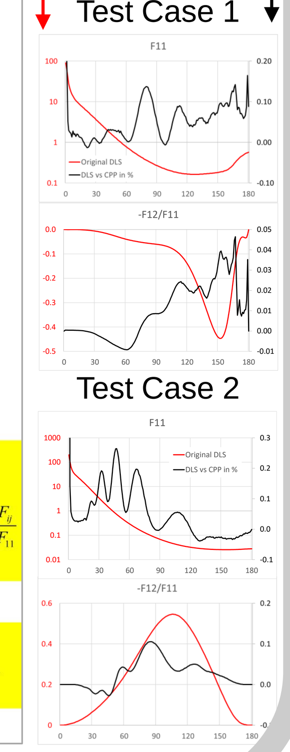
$$d_V(r) = \frac{C_V}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{\ln(r/r_m)}{2\sigma^2}\right) \rightarrow d_V(x) = \frac{C_V}{\sqrt{2\pi}\,\sigma}\exp\left(-\frac{\ln(x/x_m)}{2\sigma^2}\right);\quad C_V = \int_x d_V(x)\,d\ln x$$

- Numerical optimization (for our tasks)
  - Integrate over the size parameter $x = r / \lambda$, not r
  - Use the fixed kernels grid – drop all splines
  - Improved interpolation of "polarized" elements
  - Remove ASCII format & hard-coded parameters
  - Fixed kernels generator – must be a separate code
- Clean up the code, convert to C (phase matrix)









https://github.com/korkins/spheroids (code creating Fixed Kernels is yet to be refactored)