An implementation of SGP4 in non-singular variables using a functional paradigm

Pablo Pita Leira

15th March 2016

6th ICATT Conference, Darmstadt

Motivation

SGP4

Simplified General Perturbations 4 orbit propagator

- widely used tool for the fast, short term propagation of earth satellite orbits
- thoroughly described in the SPACETRACK report #3 by Hoots et al.
- numerous versions of SGP4: FORTRAN, C++, Java, MATLAB
- Inputs: two line elements (TLE) disseminated by NORAD

SGP4 Algorithm Description

- applied for all orbits with periods of T <= 225 min.
- secular rates of change due to the zonal harmonics J2 and J4 of the Earth potential, and due to drag perturbations in an atmosphere with a power-law altitude profile of air density
- long period corrections perturbations due to J3
- first-order, short-period perturbation corrections due to J2

SGP4Extensions: why one more version?

SGP4 is used in a broader context like conjunction analysis.

Scala can be interesting for the design of algorithms in this broader context

By having a version of SGP4 in Scala, the integration of SGP4 in the algorithms is easier.

SGP4Extensions exposes new function calls that enables new conjunction algorithms

No implementation of SDP4

SGP4Extensions

Scala

Developed by Martin Odersky in the EPFL since 2001.

- Hybrid object oriented/functional
- Rich type system
- compiled to java byte code, also possible javascript
- designed for creating DSL on top: expressive

SGP4Extensions: characteristics

- It is heavily influenced by the functional software paradigm.
- Equations have been expressed almost always literally writing the algebraic equations in the code as expressed in the papers
- Implementations using other variables and/or extra terms can be easily introduced into the propagation algorithm
- Provides more options and flexibility when being used within other algorithms, like those performing space debris conjunction analysis

Unicode support

- Unicode support to express equations
- Lyddane 2nd Order Long Period Corrections:

val $\delta I = \epsilon 3 * e \sin \omega * c$

val $\delta a = 0$

val $\delta h = -\epsilon 3 * e \cos \omega * c/s$

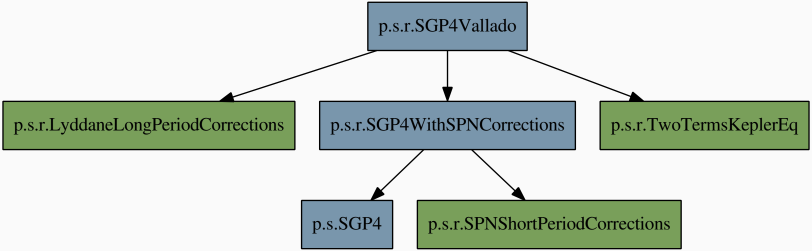
val $\delta C = -\epsilon 3 * e \cos \omega * e \sin \omega * (1/s - 2 * s)$

val $\delta S = \epsilon 3 * s - \epsilon 3 * `\eta^2` * s - 2 * \epsilon 3 * `e \cos \omega^2` * s + \epsilon 3 * `e \cos \omega^2`/s$

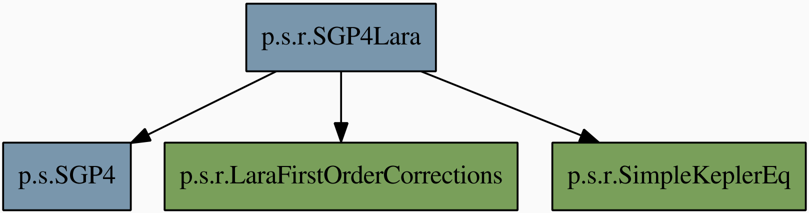
val $\delta F = \epsilon 3 * s * e \cos \omega * (1 - 2 * `\eta^2`/(1 + \eta))/2$

Note: $e \cos \omega^2$ that the product $e \cos \omega$ is squared.

SGP4 Vallado model



SGP4 Lara model



Validation results

Tested with TLEs for near Earth used by David Vallado

- SGP4Vallado match C++ results (max diffs 10^{-8}) at 30000 min
- Order of calculation of Kepler equation does not introduce significant effects
- There are small differences between other algorithms like PolarNodals, Lara and ValladoLong with SGP4Vallado

What’s next

Future's work

- Propagation of the whole catalog
- Collision analysis
- Conjuctions in field of view of optical cameras

Questions?