Readme file for paper AIAA 2006-6753

Revisiting Spacetrack Report #3

David A. Vallado[1]
Center for Space Standards and Innovation, Colorado Springs, Colorado, 80920

Paul Crawford[2]
Crawford Communications Ltd., Dundee, DD2 1EW, UK

Richard Hujsak[3]
Analytical Graphics, Inc., Exton, PA, 19341

and

T. S. Kelso[4]
Center for Space Standards and Innovation, Colorado Springs, Colorado, 80920

Current version: July 16, 2012

Based on the original NORAD 1980 Spacetrack Report #3

[1] Senior Research Astrodynamicist, Center for Space Standards and Innovation, 7150 Campus Dr, Suite 260, dvallado@centerforspace.com, AIAA Associate Fellow.
[2] Principal Engineer, 25 Blackness Avenue, pcrawford@dundee0.demon.co.uk.
[3] Orbit Determination Lead Engineer, 220 Valley Creek Blvd, rhujsak@agi.com.
[4] Senior Research Astrodynamicist, Center for Space Standards and Innovation, 7150 Campus Dr, Suite 260, tskelso@centerforspace.com, AIAA Associate Fellow.

Notes and Change Summary for AIAA-2006-6753

Revision Notes:

2011-12-30

- A user noted that the variables peo, pgho, pho, pinco, and plo in the sgp4.cmn common block are set to 0 in dscom.for. They are referenced in dpper.for (e.g., pe = pe - peo) where presumably they could have a non-zero value. The values are written out to a file by twoline2rvsgp4.for in the test program, but this wouldn't occur in other applications. Upon investigation, we noticed that these values are never set to anything other than zero. This is evident in the original versions of SGP4 as well. Thus, we concluded to leave the code as is. If we find additional developments that suggest a change is needed (ie, updates to these variables somewhere in the program), the structure will be in place to insert those additions.

2010-08-30

- Minor updates to improve performance with the exponent, or power function and low integer exponents. Some unused variables were eliminated. We received a new method to find GSTime. We did not implement this because we already have one option for improved performance using modern formulations. However, we include discussion of the change for interested parties. Essentially, this change recommended using a different form of the Julian date (no longer really a Julian date in the strictest sense, and using a 1950 year), and then finding GSTime with that. The change produces a minor effect (cm-level) on the SGP4 results. The change consists of developing a new JDay, InvJDay, and GSTime routines, shown below, and would occur mainly in the twoline2rv routine, and the initl routine where GSTime is called.

  1. In Initl, the improved operation would change

```
gsto = gstime(epoch + 2433281.5);
```

  to

```
gsto = gstime(epoch + 3382.0);
```

  2. The alternate JDay routine would have the following line different from the existing routine.

```
jd = 367.0 * (year - 1950) -
   floor((7 * (year + floor((mon + 9) / 12.0))) * 0.25) +
   floor( 275 * mon / 9.0 ) +
   day + 3382.0 +
   ((sec / 60.0 + minute) / 60.0 + hr) / 24.0;  // ut in days
```

  3. The alternate GSTime routine would use "Julian dates" from the above routine, and input them to the new GSTime routine.

```
tut1 = (jdut1 - 18263.5) / 36525.0;
```

  The important thing to remember with all of these is that the JD will no longer be a JD from 4713 BC as is traditionally understood. The InvJDay routine would also have to be adjusted for some of the outputting options in the test program.

2008-11-03

- Minor updates to improve error handling. The return codes for each routine were switched from Integer to Boolean. This permits a simple check of runtime success, while the details of the error are contained in the error integer value.

2008-09-03

- New version (and version identifier included in the code) to coincide with the differential correction program. Minor updates for eccentricity checks and tolerances. Changes for the DSPACE routine and the integrator to enhance computational speed for certain satellites. Some of the zip files were missing some files and Matlab files had upper and lower case filenames. An option was added to permit two modes of operation – one that emulates what we believe AFSPC does, and another that uses improved processing techniques that AFSPC may not be using.

November 2007

- Updates to make the application more compatible with the AFPSC implementation based on user inputs. Specifically, the GHA (sidereal time) was set back to the older way of processing. This introduces sub-mm differences. Also, there were a few operations where the original code had used user written functions (mod, atan2). These functions returned certain quadrants and in some places in the code, the results were used without a trigonometric argument. Thus, the quadrant became important for those cases to matching the same answers. We did not resurrect the older approaches, instead because there were only a couple of instances of this, simple conditional statements were inserted to simulate the intended behavior.

April 2007

- Updates for manual operation. The program uses a verification mode (with additional parameters at the end of the second line in the .tle file containing the start/stop/delta minutes from epoch (mfe) information) to permit quick checks of relevant times and spacing of ephemeris generation. The catalog mode lets the user test an entire satellite catalog for a two day span centered on the epoch time of each element set. The manual operation gives three additional options – entering the start/stop/delta mfe information, entering the YMDHMS information for the start and stop times, or entering the year and dayofyr for the start and stop times. This should give enough flexibility to permit various uses for the tool.
- Constants were re-worked to use the machine precision available. For instance, pi, twopi, rad2deg, deg2rad, etc. can be calculated from pi, which is defined in some languages. Where it is not, 4.0*atan(1.0) is used. In addition, mathematical and astrodynamic constants were separated.
- Misc formatting to correct typos, misspellings, etc.
- Removed the “ildm” variable in DPPER as it was not being used. It was originally used during development of the 2006 paper as a Lyddane choice switch, but had later been made obsolete.

August 2006

- Original version baseline for the paper

Operation Notes:

There are several variables that can be set for operation in each of the programming languages. These are designed to give some flexibility in the runs and provide some examples as you use the program.

To verify your installation, execute the testmat (testcpp, etc) script in Matlab (c++, etc).

1. At the input mode prompt, enter "a" for the afspc mode. The improved mode is included to provide the structure should you want to try other options in the code.
2. At the input type prompt, enter "v" for the verification mode. This mode will run through a series of TLE's and produce the standard output. It can then be compared with the original output to ensure the program is working properly. Another option is "c" for a catalog mode where the program automatically calculates an ephemeris for each TLE over a period of time. We used this for the whole satellite catalog tests while we were working on the paper. Finally, there is a manual mode "m" where the user can enter the start/stop times, or the minutes from epoch. Notice that the TLE files for the first option has additional information appended to the first line of the TLE. This is so the program can read the specific times to perform the test. You will not find these additional times in routine TLE's obtained from the web.
3. At the input constants prompt, enter "72". This is specifies the WGS-72 constants that are used by SGP4. We had done some tests with updated constants (WGS-84 (84), and a low precision WGS-72 (721)), but these do not appear to be used operationally. The option is included so the structure is in place should there be a change to the SGP4 operation.
4. At the input elset prompt, enter "sgp4-ver.tle". This is the test case of TLE's. There are some that test reentry and other conditions, so a couple of error messages appearing to the screen are normal and you can see these in the .out file.

The elset numbers will scroll down the screen and some error messages will appear. When execution is complete, do a file comparison of your output file "tmatver.out" with "tmatvera.out" from the zip file.

The output will be an ephemeris with time tags. Some languages have orbital elements, others do not. The coordinate system is the TEME frame described in the paper. Conversion to other frames is best handled by converting TEME to ECEF (ITRF). From ECEF, standard coordinate transformation routines can be used. Note that leap seconds are not accounted for by SGP4. The mathematical theory has no place for this processing as it is an output option. Any ephemerides spanning a leap second occurrence should be addressed by other software. The various languages may have slightly different answers, but these are all at the sub-meter level.

Language Notes:

Each language has files (debug1.m, debug1.cpp, debug1.for, debug1.pas) to facilitate debugging the values as a program executes. They are disabled in the codes as shipped, but may be enabled if needed.

C++

The C++ executable needs a Borland C++ specific DLL to run (cc3260mt.dll). If you do not have Borland C++, simply recompile the code with your specific compiler.
Lee Barker has prepared a more object oriented version of the C++ code.
Contents:

testcpp.cpp	Main program for example use
sgp4unit.cpp (.h)	sgp4 mathematical theory
sgp4io.cpp (.h)	I/O routine for TLE data (twoline2rv conversion)
sgp4ext.cpp (.h)	Additional routines needed for the test program (jday, rv2coe, etc)
debug*.cpp	Debug routines to print out intermediate variables at the end of each function call

FORTRAN

Contents:

testfor.for	Main program for example use
sgp4unit.for	sgp4 mathematical theory
sgp4io.for	I/O routine for TLE data (twoline2rv conversion)
sgp4ext.for	Additional routines needed for the test program (jday, rv2coe, etc)
astmath.cmn	Math constants (not a common in the usual sense)
sgp4.cmn	sgp4 common variables (near earth and deep space)
lksgp4.bat	makefile for Lahey FORTRAN compiler/linker
debug*.for	Debug routines to print out intermediate variables at the end of each function call

Java

Joe Coughlin and Kurt Motekew have converted the code to Java. It has not been fully tested against these versions yet.

Matlab

The Matlab version is due to Jeff Beck who performed the original translation. He provides the following notes. The code is a line-by-line translation of Vallado's C++ version of 28 Jun 05. The files are grouped below according to the “unit” “io” “ext” files in the other languages.
Matt Schmunk has provided a vectorized form of Matlab that should run significantly faster. It has not been fully tested against the other approaches yet.
Contents:

testmat.m	Driver script for testing and example usage
sgp4.m	Main sgp4 routine
sgp4init.m	Initialization routine for sgp4
initl.m	Initialization for sgp4
dsinit.m	Deep space initialization
dspace.m	Deep space perturbations
dpper.m	Deep Space periodics
dscom.m	Deep Space common variables
twoline2rv.m	TLE conversion routine
angl.m	Find the angle between two vectors
constmath.m	set mathematical constants
days2mdh.m	convert days to month day hour minute second
getgravc.m	Get the gravity constants
gstime.m	Find Greenwich sidereal time
invjday.m	Inverse Julian Date
jday.m	Find the Julian Date
mag.m	Magnitude of a vector
newtonnnu.m	Kepler’s iteration given eccentricity and true anomaly
rv2coe.m	Convert position and velocity vectors to classical orbital elements
debug*.m	Debug routines to print out intermediate variables at the end of each function call

To generate debug output, execute the following lines in Matlab before executing sgp4test:

```
global idebug
idebug = 1;
```

If you have any corrections, comments, or suggestions, please feel free to contact me at beckja@alumni.lehigh.edu. Also, if you develop any supplemental routines (e.g. a GUI driver or an orbit display) and would like to share them, I'll be happy to include them with future versions.

Jeff Beck, beckja@alumni.lehigh.edu, 19 Oct 2005

Pascal

The Pascal code was developed using Borland's Turbo Pascal. As such, it should be compatible with most Pascal compilers, but there may be some I/O routines that will be different. The underlying mathematical operations should be consistent.

The NewDelay unit is needed for some older DOS applications that try to use Turbo Pascal on faster computers. Essentially, NewDelay slows the processing down to permit operation on modern computers.

The Borland extended type is advertised as a 10-byte solution whereas the other codes are all 8-byte operations.

Contents:

testpas.pas	Main program for example use
sgp4unit.pas	sgp4 mathematical theory
sgp4io.pas	I/O routine for TLE data (twoline2rv conversion)
sgp4ext.pas	Additional routines needed for the test program (jday, rv2coe, etc)
debug*.pas	Debug routines to print out intermediate variables at the end of each function call