Test Plan for LSST Cadence Simulator (OpSim) Release 2.1

Michelle Miller, Cathy Petry, Francisco Delgado

1 Introduction

1.1 Overview of Simulator

The Operations Simulator (OpSim) for the Large Synoptic Survey Telescope (LSST) is a tool to explore the design and operations parameter space of the LSST ten-year mission to perform concurrent scientific surveys. The design is modular, with separate science programs being described in separate modules. There are sophisticated telescope and sky models. All important parameters for the telescope, the site, and the science programs are easily accessible in configuration files. OpSim uses seeing and cloud data from the three potential LSST sites for our simulations.

1.2 Test Philosophy/Strategy

Testing the simulator will happen in three phases. The first two phases will focus on software validation, while the third phase will verify scientific algorithms. First the simulator software must be validated. To meet this goal, a series of unit tests will be created. These tests will form the basis of a regression test suite that can be run after major software changes or prior to a new release of the code. After validating the software units, a small number of integration tests will be performed to ensure the validity of the software that uses the low-level tasks. Once the software has been validated, a number of science verification tests will be completed to assess the scientific accuracy of the simulation. A variety of datasets will be created to cover differences in the site, weather, set of proposals, and field selection. All science verification tests will consist of post-processing these datasets.

To completely cover the LSST cadence parameter space of science proposal, site, and observational path across the sky, we have created a 3-dimensional grid to track each permutation. We have a series of tests to be run to ensure the scientific accuracy of the LSST Cadence Simulator (OpSim). Not every test needs to be run for every variation. Further, we have assumed that the 450 parameters that can change simulator behavior are in a fixed configuration based on Kem Cook's "golden" values while we prove in the accuracy of the models and scientific algorithms. The efficacy of achieving the science goals will be addressed in a future paper. Scientists can tweak the behavior of the simulator by finding parameter values they like once we have "certified" the software.

2 Software Unit Tests

Each software module contains many functions. Unit tests will validate the functions within each module or object. These tests are of the form input in, expected output out. Does the actual output match the expected output? The tests can be implemented as Python unit tests.

2.1 Astronomical Sky module

This module contains astronomical calculations relating to sky conditions for a given date/time and site (longitude/latitude).

2.1.1 getTwilightSunriseSunset()

Choose ten to fifteen dates (year, month, day) and specify the longitude/latitude coordinates for several sites. One of the chosen dates must be the first day of simulation since it is a problematic boundary case. For a fixed epoch, verify that the sunrise and sunset times for that date are correctly calculated by getTwilightSunriseSunset() for astronomical twilight in MJD and seconds since start of simulation.

2.1.2 getMoonPhase()

Choose ten to fifteen dates in MJD format. Calculate sun and moon position for the given MJD. Derive moon phase as a percentage from 0 to 100 based on these positions. Verify that getMoonPhase() calculates the same number.

2.1.3 getSkyBrightness()

For a given location, choose a set of fields at differing times and differing extinction. Compare the result of getSkyBrightness() for these field/time/extinction sets with hand-calculated values.

2.1.4 getHAforAirmass()

Choose ten to fifteen airmass quantities for two different site latitudes. Calculate (either by hand or using a package other than slalib) positive hour angle (setting fields) for given airmass.

2.1.5 airmass()

Choose ten to fifteen dates in elapsed simulation seconds and specify the longitude/latitude coordinates for several sites. Compute local sidereal time in radians and MJD. Choose several visible fields for that date at the chosen longitude/latitude. Calculate airmass at that zenith angle for each field. Compare these values with those computed by the airmass() function.

2.1.6 getPlanetPosition() – not used

Choose several dates (seconds of simulation time), one or more planets, and one or more sites. Compute MJD for this date using standard epoch. Determine the RA and Dec (radians) of the planet given the MJD and longitude and latitude of the site. Compare these values with those returned by the getPlanetPositon() function.

2.1.7 getDistance() – not used

Choose several pairs of coordinates (RA/Dec) and calculate the distance between them. Compare results with those calculated by getDistance().

2.2 Filters

2.2.1 computeFilterSeeing()

Choose a grid of 5 seeing values and 5 airmass values, and compare the obtained value of the corrected seeing for each filter against the reference values.

2.2.2 allowedFiltersForBrightness()

Given a set of configuration values for filters sky brightness limits, choose a list of 30 sky brightness values and compare the obtained list of allowed filters in each case to the reference list.

2.3 Instrument

For the unit tests for the Instrument component (which deals with the dome, telescope and camera models) there are three important considerations.
The first one is the configuration of the models used for the test cases. A unique set of parameters must be chosen for the tests, namely the observatory location and the slew factors. This set is not necessarily the values for the corresponding version of the simulator, they could be modified to clarify the test results.
The second one is that for some complex units there is the need of an initial state that is not given in the method's arguments, they are built from the history of movements in the model. To achieve this, the unit test needs to be performed by a specially written driver in replacement of the scheduler component. This is important because for some telescope positions there is more than one possible state due to the cable wraps.
The third one is the reference values for some of the tests. In the complex units (GetSlewDelay, Slew, Observe) the result is a set of tables, very hard to manually calculate. So the proposed approach for these cases is to obtain the tables from the unit tests, and validate them manually. After that the result can be labeled as the reference.

The telescope state is composed of: Dome Alt Az coordinates, Mount Alt and absolute Az (>360 in range), Rotator absolute angle, filter position, tracking/no-tracking state, time at which the current tracking started, RA Dec coordinates if it is tracking.
The DB tables generated in the “Observe” method give the complete state for the telescope before and after the slew, as well as the slew activities delays, maximum speeds and critical path.

2.3.1 RaDec2AltAz()

Choose a list of 30 Ra-Dec-time coordinates, and compare the obtained Alt-Az positions against the reference list.

2.3.2 AltAz2RaDec()

Choose a list of 30 Alt-Az-time coordinates, and compare the obtained Ra-Dec positions against the reference list.

2.3.3 GetShortestDistanceWithWrap()

Choose absolute limits for minimum and maximum angle. Choose a grid of 5 absolute initial angles (between the limits) and 5 target angles, and verify that for each case the obtained absolute final angle is correct and inside the limits, and the distance obtained is the shortest option.

2.3.4 TimeAccelMove()

Choose a maximum speed, an acceleration, and a deceleration. Choose 30 target distances to move (some positive, some negative), and compare the obtained movement time and maximum reached speed against the reference values.

2.3.5 UpdateState()

Choose a list of 5 initial telescope states and 5 time interval, and compare the obtained final state in each case against the reference final states after tracking for the respective elapsed time.

2.3.6 SetClosestState()

Choose a grid of 5 initial telescope states and 5 target positions, and compare the obtained state against the reference list of closest telescope states that satisfy the target in each case.

2.3.7 GetDelayForDomAlt()

Choose a grid of 5 initial and 5 target Dome Altitude positions, and compare the obtained delay times against the reference values.

2.3.8 GetDelayForDomAz()

Choose a grid of 5 initial and 5 target Dome Azimuth positions, and compare the obtained delay times against the reference values.

2.3.9 GetDelayForTelAlt()

Choose a grid of 5 initial and 5 target Telescope Altitude positions, and compare the obtained delay times against the reference values.

2.3.10 GetDelayForTelAz()

Choose a grid of 5 initial and 5 target Telescope Azimuth absolute positions, and compare the obtained delay times against the reference values.

2.3.11 GetDelayForTelOpticsOL()

Choose a grid of 5 initial and 5 target Telescope Altitude positions, and compare the obtained delay times against the reference values.

2.3.12 GetDelayForTelOpticsCL()

Choose a grid of 5 initial and 5 target Telescope Altitude positions covering the limits for the closed loop optics correction in the configuration parameters, and compare the obtained delay times against the reference values.

2.3.13 GetDelayForRotator()

Choose a grid of 5 initial and 5 target Rotator angle positions, and compare the obtained delay times against the reference values.

2.3.14 GetDelayForFilter()

Choose a grid of 5 initial and 5 target filters, and compare the obtained delay times against the reference values.

2.3.15 GetDelayForTarget()

Given a telescope state, choose a list of 30 targets and compare the obtained delay times against the reference values.

2.3.16 GetSlewDelay()

Choose a grid of 5 telescope states and 5 target positions, including cases that test cable wrap limits, fields too close to zenith or below horizon. Compare the obtained delay time with expected results.

2.3.17 GetCurrentTelescopePosition()

Choose a grid of 5 telescope states and 5 times, and compare the obtained sky position for every combination against the reference positions.

2.4 Simulator

2.4.1 computeDateProfile()

Choose five to ten dates (seconds of simulation time) and one or more sites. Compute MJD for this date using standard epoch. Compute local sidereal time for this date at this longitude. Compare these quantities with those generated by computeDateProfile().

2.4.2 initMoonPhase()

Choose several dates (seconds of simulation time) and one or more sites. Compute MJD for this date using standard epoch. Calculate the moon phase for tonight and last night for each date. Compare values to determine whether the moon is waxing or waning. Compare results with initMoonPhase().

2.4.3 computeMoonProfile()

Choose several dates (MJDs) and one or more sites. Calculate the moon RA and Dec for the site and date. Calculate the moon phase for each date. Compare values with the results of computeMoonProfile().

2.4.4 startNight()

Choose several dates (seconds of simulation time) and one or more sites. Determine if moon is waxing or waning. Is this a new lunation? Is this a new year?

2.4.5 setupSimulation() – integration test (instantiate objects)

This function instantiates Instrument, Filters, AstronomicalSky, Weather, ObsScheduler, NonObsScheduler, TAC, and Operations objects. Not sure how to test this function other than making sure that each of these objects exists after the function exits.

2.4.6 start() – integration test (activate Simulator)

This function activates simulation objects TAC and Operations. It then aligns the simulation timer to night time, determining the next sunrise and sunset times in simulation seconds and MJD. It determines the moon phase and starts the first night. We then enter a scheduling loop that schedules observing and downtime events for the duration of the simulation. Both ObsScheduler and NonObsScheduler objects are asked for their top observation or downtime event, respectively. This method then compares the ranks of both events and grants the time to one of them. All timers are incremented by the amount of time the simulated event would have used and the event is recorded in the database (either in ObsHistory or NOBHist). Next, this function recalculates the next sunrise and sunset and starts a new night if the event duration is greater than one day. Otherwise, it tests whether or not it is still night. If we have crossed the time for sunrise, then park the telescope and wait until sunset. Start a new night of observing and calculate the next sunrise and sunset.

Is the start of night calculated correctly? Given a downtime event that is greater than one day, are sunrise and sunset calculated correctly?

2.5 Weather

For purposes of testing, we assume the weather model provided by Charles Claver is correct. We do not test it here. In addition, the two functions that query the database to determine the amount of weather data are trivial and will not be unit tested. These are loadDaysofCloudData() and loadDaysOfSeeingData().

2.5.1 getSeeing()

Choose several dates (seconds of simulation time) and one or more sites. Find closest time in Seeing DB table for site and record the seeing value. Compare with getSeeing() results. Should we be interpolating values? How close are the values?

2.5.2 getTransparency()

Choose several dates (seconds of simulation time) and one or more sites. Find closest time in Cloud DB table for site and record the cloud value. Compare with getTransparency() results. Should we be interpolating values? How close are the values?

3 Software Integration Testing

Software integration testing focuses at the level of putting pieces (or modules) together to form larger functional units. Although these amalgamated pieces are more functional that just modules, they have fewer capabilities than the entire system. The point is to find

problems with module interaction that would be harder to locate if we put the whole system together before testing the interactions.

3.1 Recording results in Database

During a simulation, every quantity used for a calculation including time is stored in the database. This must be the case if we hope to have repeatable results. Someone should be able to take the quantities in the database and derive by hand the results we have obtained. Also, the database needs to allow for data mining based on how the simulator was configured.

3.1.1 Simulation Configuration

Is the version number of the simulator stored with the results (session table)? Are all configuration parameters from *.conf files stored correctly in the database?

3.1.2 Observations

Check types to make sure that the value stored matches the in-core calculation type. Print contents of many (twenty) observations before they are stored to ObsHistory. Compare with values in the database. Make sure they are being stored correctly.

3.2 Fields

Generate the fields with the corresponding scripts, and populate the Fields table in the database. Then validate the field generation as given below.

3.2.1 Check minimum distance

For each field, find the closest neighbor and check that the distance is compatible with the FOV.

3.2.2 Check sky coverage

Build a plot of the field centers and check that all the sky is covered.

3.3 Scheduling Observations

3.3.1 Scheduler Functional Description

Given a date, site, and telescope position, setup the scheduler, add and activate all proposals, and start the night. For a given date (seconds of simulation time), get moon, cloud, and seeing information. Calculate the target profile for each field in the list of targets. For each field, calculate the airmass, altitude of the field in the sky in radians, distance to the moon, the moon's altitude, and sky brightness. Also determine a list of filters that could be used for the field given the sky brightness, seeing, and airmass. Build a proximity array between current telescope position and all fields under consideration, as specified by each proposal in updateTargetList(). Ask each proposal to suggest its top N observations, where N is specified by *numSuggestedObsPerProposal* in Scheduler.conf. Update target list with ranks from proposals where the new rank is added to the existing rank if the field is already stored in the list from a different or the same proposal. The

parameter *reuseRankingCount* from Scheduler.conf determines the number of observations to be taken using calculations (such as slew time, rank, seeing, transparency) at a single time.

3.3.2 Testing Scheduler

Using stubs of the proposals that are programmed to send a list of observations based on date, site and telescope position, we can validate the scheduler logic that winnows the set of proposed observations to one. The idea is to test the scheduling algorithms in isolation from the Instrument and the Proposal modules. We may adjust the parameters, *numSuggestedObsPerProposal* and *reuseRankingCount*, to assess their effect on the scheduling process.

3.4 Integrated Instrument

These functions are more encompassing and are composed of many small functions brought together to act in a coordinated fashion. As such, they should be treated as integrated pieces.

3.4.1 Slew()

Choose a grid of 5 telescope states and 5 target positions, including cases that test cable wrap limits. Compare the obtained telescope delay time, final state, axis maximum speeds and slew activities details for each combination against the reference tables.

3.4.2 Park()

Given a telescope state, perform a Park followed by an Observe, and compare the intermediate telescope states from the DB tables against the reference.

3.4.3 Observe()

Given a telescope initial state, choose a list of 30 targets and observe the whole sequence. Compare every intermediate state, slew activities and delay time from the DB tables against the precalculated reference.

3.5 Instrument/Scheduler Interaction

Instrument provides a slew time, Instrument.getDelayForTarget(), to the Scheduler for each field suggested by the proposals. A slew factor gives a weighting to the slew time for tradeoffs between minimizing slew and going to fields that could be far away, but must be observed soon. The Instrument also enforces limits on telescope movement, such as rotator and cable-wrap. These tests focus on the interaction between the Instrument model and the Scheduler. A proposal stub will suggest a certain number of fields with the same numeric rank, so the scheduler will select the best target using only the slew time contribution.

3.5.1 Simple Slew Time Factor in Scheduling

Given a date, site and telescope initial position, choose 30 fields close to the telescope's initial position that will be up for the duration of the simulation. To simplify, we will limit the observing to one filter. The proposal stub will give all fields the same rank. We

expect to see the field closest to the telescope's initial position be observed first, with each subsequent observation being the next closest one to our current position.

3.5.2 Physical Constraints of Instrument and Scheduling

Given date, site and telescope initial position, choose 30 fields far enough apart to ensure that the Instrument hits its cable wrap limit. To simplify, we will limit the observing to one filter. The proposal stub will give all fields the same rank. We expect to see the field closest to the telescope's initial position be observed first, with each subsequent observation being the next closest one to our current position until we reach the field that will require a long slew because of the cable wrap limit.

The expected result should be multiple consecutive observations for the same field-filter (minimum slew-time) until the azimuth axis in the mount reaches the cable wrap limit, the rotator tracking limit is reached or the field is no longer visible.

3.5.3 Scheduling Timed Sequences Given Instrument Physical Constraints

Choose a sky region covering 30 fields. Build a test proposal that suggests the same ranking for all the field-filter combinations except for the field-filters already observed, this is, it requires 1 observation for each field-filter combination. Analyze the sequence of observations obtained in a run. The result should be long sequences of observations of consecutive fields on sky, except for conditions in which the mount azimuth cable wrap limit is reached, the rotator tracking limit is reached or the telescope reached a sky airmass limit.

3.6 Science Proposals

These unit tests are actually implemented as more integrated tests, and they need a special scheduler/driver to orchestrate the actions and states. The objective of the "suggestObservation" tests for the 3 different type of proposals considered, is to determine if each proposal is implementing the rank algorithms correctly. The validation of the algorithms in the context of pursuing the scientific goals is better suited in an integral test case.

To test the unit "suggestObservation" it is necessary to modify the status of the accounted observations in the proposal, to see the effect of those conditions into the ranked list obtained. An easy way to modify the internal history of the proposal is to simply handle the proposal as if the system were actually performing the observations.

3.6.1 WeakLensingProp.updateTargetList()

Choose 10 dates and check if the plot of the obtained list of targets corresponds to the visible sky during the night minus the galactic bulge.

3.6.2 NearEarthProp.updateTargetList()

Choose 10 dates and check if the plot of the obtained list of targets corresponds to the ecliptic zone for the visible sky during the night.

3.6.3 SuperNovaSubSeqProp.updateTargetList()

Choose 10 dates and check if the plot of the obtained list of targets corresponds to the visible sky during the night minus the galactic bulge. Also check if the restricted list of targets for the creation of new sequences corresponds to the specified East-West limits.

3.6.4 suggestObservation()

For each of the proposals, WeakLensing, NearEarth and SuperNovaSubSeq, choose a sequence of 30 test-targets, and with a special scheduler performs the observations analyzing during the process the rank values delivered by this unit.
-choose a date and configuration and call updateTargetList().
-consider this obtained target list for the 30 test-targets, also adding some invalid targets.
-for each test-target choose a correlative observation test-time during the test night.

- call suggestObservation() with the test-time and verifies that the ranks follow the algorithm.
- Call closeObservation() with the test-target and continue the loop.

4 Science Verification Tests

4.1 Generating Test Datasets

We will try to simplify potential interactions in the models by creating a number of datasets that range from excessively simplistic to realistic. The first datasets will be rudimentary, and complexity will be added bit by bit. Our attempt is to limit data size and to create predictable patterns in the data while we verify scientific accuracy. A variety of datasets will be created to cover differences in site, weather, set of proposals, and field selection (see Figure 1).

4.1.1 Field Selection

There are three field selection permutations: 6 equatorial single fields, 6 equatorial field pairs, and the complete all sky field catalog.

Six Equatorial Single Fields

These fields are chosen, somewhat arbitrarily, to be evenly spaced along the celestial equator. This simple set of fields is designed to test very basic properties of the Simulator.

R.A.	Dec.
00h 00m 00s	00d 00m 00s
04h 00m 00s	00d 00m 00s
08h 00m 00s	00d 00m 00s
12h 00m 00s	00d 00m 00s
16h 00m 00s	00d 00m 00s
20h 00m 00s	00d 00m 00s

Six Equatorial Field Pairs

These fields are chosen to be evenly spaced along the celestial equator, and offset from precisely zero hours in Right Ascension to test non-zero positions. Declinations are chosen so that one field width (3.5 degrees) separates the pair, and so that the pair straddles the equator in order to test positive as well as negative declinations.

R.A.	Dec. 1	Dec. 2
02h 35m 45s	+06d 54m 30s	-07d 05m 30s
06h 35m 45s	+06d 54m 30s	-07d 05m 30s
10h 35m 45s	+06d 54m 30s	-07d 05m 30s
14h 35m 45s	+06d 54m 30s	-07d 05m 30s
18h 35m 45s	+06d 54m 30s	-07d 05m 30s
22h 35m 45s	+06d 54m 30s	-07d 05m 30s

All Sky

This is the full catalog of field positions generated by Kem Cook's algorithm.

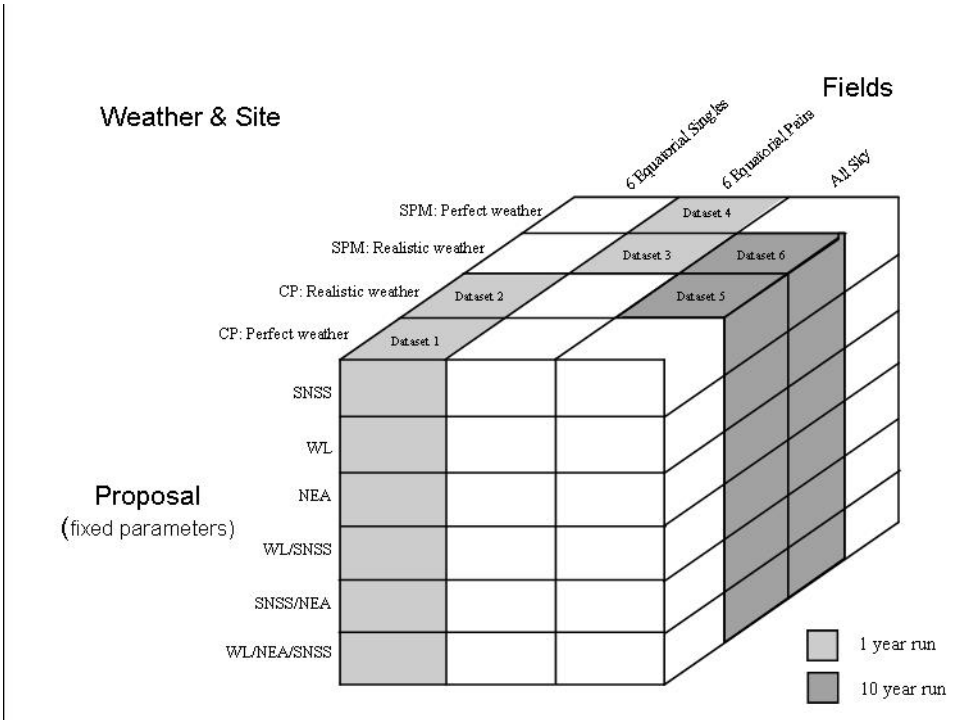

Figure 1.

4.1.2 Dataset #1

These datasets will be based on perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments. They will simulate one year of observing.

- **Dataset 1.1**: SNSS proposal run alone using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.
- **Dataset 1.2**: Weak Lensing proposal run alone using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.
- **Dataset 1.3**: Near Earth Asteroid proposal run alone using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.
- **Dataset 1.4:** Weak Lensing and SNSS proposals run together using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.
- **Dataset 1.5:** Weak Lensing and Near Earth Asteroid proposals run together using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.
- **Dataset 1.6:** SNSS and Near Earth Asteroid proposals run together using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.
- **Dataset 1.7:** Weak Lensing, Near Earth Asteroid, and SNSS proposals run together using perfect weather and seeing on Cerro Pachón and a grid of single fields on the sky with the same Dec and RA varying in 4 hour increments.

4.1.3 Dataset #2

These datasets will be based on a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec and the RA varying in 4 hour increments. The datasets will simulate one year of the survey.

- **Dataset 2.1:** SNSS proposal run alone using a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with same Dec and RA varying in 4 hour increments.
- **Dataset 2.2:** Weak Lensing proposal run alone using a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments.
- **Dataset 2.3:** Near Earth Asteroid proposal run alone using a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments.
- **Dataset 2.4:** Weak Lensing and SNSS proposals run together using a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments.

- **Dataset 2.5:** Weak Lensing and Near Earth Asteroid proposals run together using a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments.
- **Dataset 2.6:** SNSS and Near Earth Asteroid proposals run together using realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments.
- **Dataset 2.7:** Weak Lensing, Near Earth Asteroid, and SNSS proposals run together using a realistic weather and seeing model for Cerro Pachón and a grid of single fields on the sky with the same Dec. and RA varying in 4 hour increments.

4.1.4 Dataset #3

These datasets will be based on perfect weather and seeing on San Pedro Mártir and a grid of pairs of fields on the sky with the similar Dec. and RA varying in 4 hour increments. The datasets will simulate one year of observing.

- **Dataset 3.1**: SNSS proposal run alone using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 3.2**: Weak Lensing proposal run alone using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 3.3**: Near Earth Asteroid proposal run alone using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 3.4:** Weak Lensing and SNSS proposals run together using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 3.5:** Weak Lensing and Near Earth Asteroid proposals run together using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 3.6:** SNSS and Near Earth Asteroid proposals run together using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 3.7:** Weak Lensing, Near Earth Asteroid, and SNSS proposals run together using perfect weather and seeing on San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.

4.1.5 Dataset #4

These datasets will be based on a realistic weather and seeing model for San Pedro Mártir and a grid of pairs of fields on the sky with the similar Dec. and RA varying in 4 hour increments. The datasets will simulate one year of observing.

- **Dataset 4.1:** SNSS proposal run alone using a realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.

- **Dataset 4.2:** Weak Lensing proposal run alone using a realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 4.3:** Near Earth Asteroid proposal run alone using a realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 4.4:** Weak Lensing and SNSS proposals run together using a realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 4.5:** Weak Lensing and Near Earth Asteroid proposals run together using a realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 4.6:** SNSS and Near Earth Asteroid proposals run together using realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.
- **Dataset 4.7:** Weak Lensing, Near Earth Asteroid, and SNSS proposals run together using a realistic weather and seeing model for San Pedro Mártir and a grid of field-pairs on the sky with similar Dec. and RA varying in 4 hour increments.

4.1.6 Dataset #5

This dataset will be based on a realistic weather and seeing model for Cerro Pachón using the full visible sky with no overlaps. The dataset will simulate ten years of observing.

- **Dataset 5.1:** Weak Lensing and Near Earth Asteroid proposals run together using a realistic weather and seeing model for Cerro Pachón and the full visible sky.

4.1.7 Dataset #6

This dataset will be based on a realistic weather and seeing model for San Pedro Mártir using the full visible sky with no overlaps. The dataset will simulate ten years of observing. One ten-year run with the three proposals enabled should be sufficient for testing.

- **Dataset 6.1:** Weak Lensing, Near Earth Asteroid, and SNSS proposals run together using a realistic weather and seeing model for San Pedro Mártir and the full visible sky.

4.2 Test case #1: Verify time of observation

Are the fields observed at night (astronomical twilight)? Check this for each site and for each science program that has different twilight definitions.

- Compute the local time of observation for the site for which data is being plotted
- Compute the local times of twilight for the site for which data is being plotted (need elevation, longitude, latitude)
- Plot:

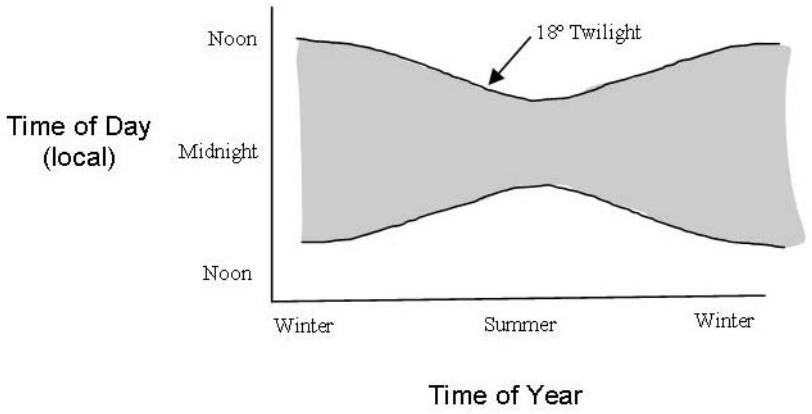

Figure 2.

- Observations are expected to fall inside the shaded regions and not outside this region at all

What is the minimum zenith angle at the time of observation? Is this value acceptable given there is no atmospheric corrector in place? Is the minimum zenith angle a specification that is proposal dependent?

- Compute the sun's altitude (need date) and field altitude or zenith angle (need RA, Dec., date)
- Plot:

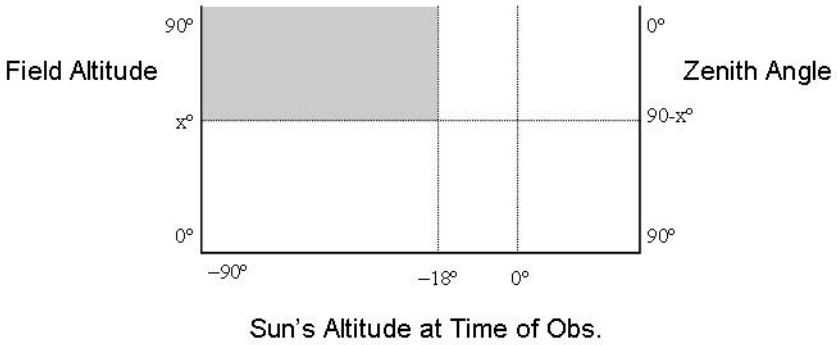

Figure 3.

- All field positions are expected to fall in the shaded region.

- A horizontal limit at x degrees is expected to define a zenith angle (and corresponding airmass) limit for the observations.
- The minimum zenith angle (or maximum airmass) is expected to be consistent with the constraint imposed by each proposal.

Does the zenith angle correlate with airmass in a way that is in agreement with the known relationship for all observations?

- Plot airmass at observation vs. zenith angle for all fields:

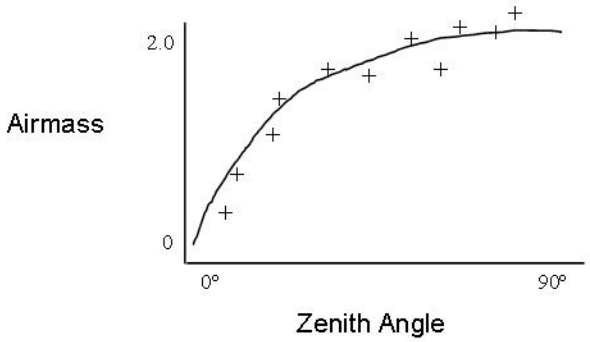

Figure 4.

- The region covered by the data points are expected to be well fit by a known relationship.

4.3 Test case #2: Verify zone of avoidance

Are the observations outside of the zone of avoidance for the given time period?

- On a sky projection, plot the observed field centers (and/or "patches") with respect to the Galactic plane.
- Overplot lines designating the "zone of avoidance" given in the simulator.

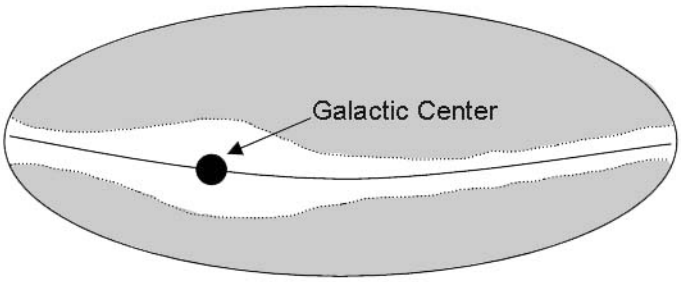

Figure 5.

- Field centers are expected to lie outside the designated "zone of avoidance", enclosed by the shaded region.
- This plot is expected to vary with proposal.

4.4 Test case #3: Verify sky brightness calculation

Is the Moon being avoided? By how much? What is the filter dependency?

- Compute the Moon's position (R.A., Dec. from ephemeris & date), its altitude, phase, and the angular distance from the Moon to the field center.
- Count how many objects are observed when no moon is up, i.e. altitude ≤ 0.
- For fields observed when the Moon's altitude is > 0, plot for each filter:

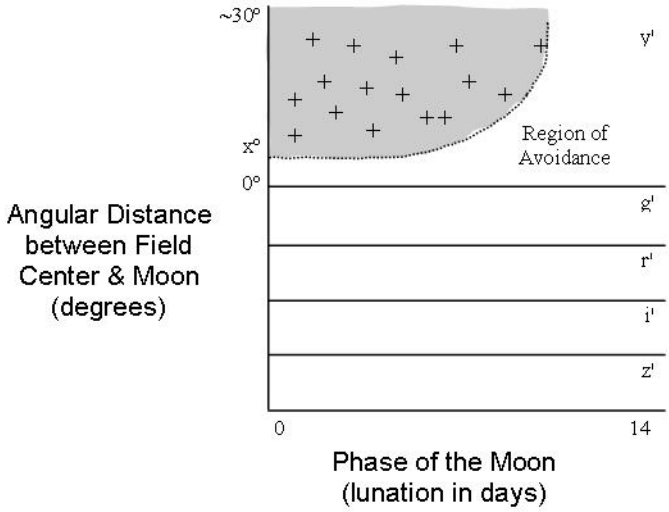

Figure 6.

- The y-intercept, x°, is some minimum distance from the Moon permitted by the Simulator.
- The data points are expected to fill the shaded region (for each filter), and create a "region of avoidance" that delineates the limit of sky brightness to which the Simulator will allow observations to take place.

Is the sky brightness (in each filter) calculated correctly?

- Compare the sky brightness derived by the Simulator (found in ObsHist), the value computed by numerical means using radiative transfer (Philip Pinto's code), and the observed value from SkyCam.
- For each observation of a field in a particular filter where the Moon's altitude is greater than zero (i.e., the Moon has risen), obtain the three sky brightness values described in the previous bulleted item.
- Plot:

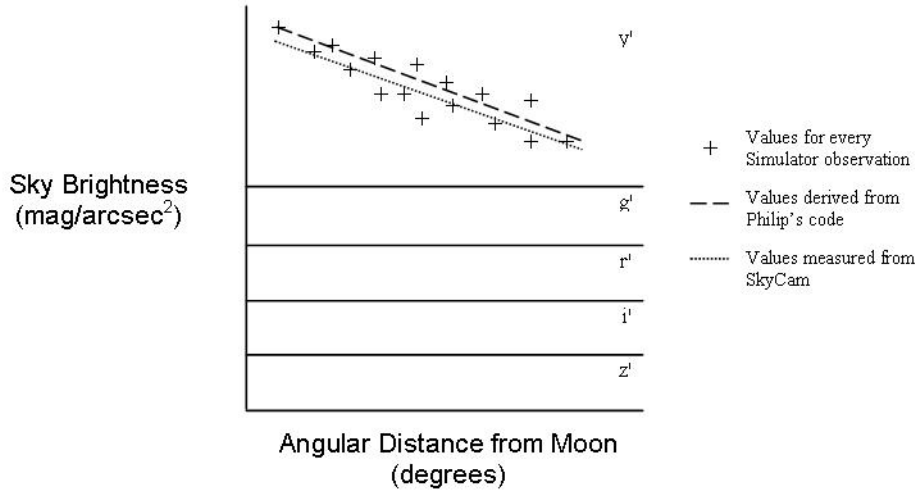

Figure 7.

- Good agreement is expected to found among these three quantities.

Is an appropriate filter chosen for the observation of a field given the current sky brightness conditions at the location of the field? Note that these results are expected to vary with proposal.

- For each filter, plot the distribution of sky brightness at the time of observation:

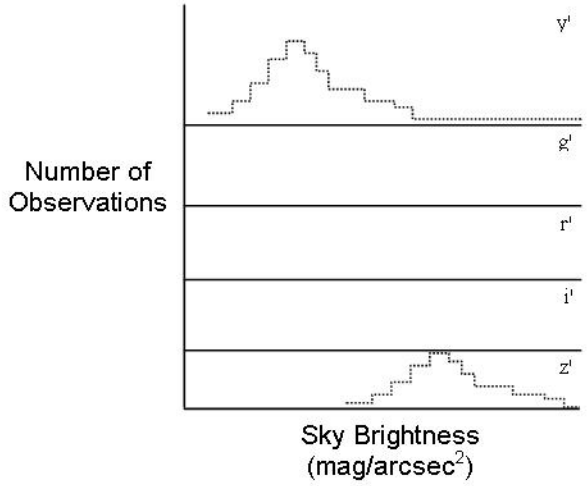

Figure 8.

- The distribution for redder filters is expected to be skewed towards more bright-sky observations.

4.5 Test case #4: Verify observing efficiency

How is the total time available for observing spent? (pie plot?)

- Of the time available for observing (good weather and seeing), what percentage is spent actually observing?
- What percentage of time is spent slewing or on filter changes?
- Is there a significant amount of other deadtime? What is the source (weather, other)? And is this as expected?
- Is the mechanical downtime model realistic?

How is the time between observations spent?

- What is the distribution of times between observations?
- What is the distribution of slew distances?
- Given the competing science programs and weather, is the Simulator observing adjacent fields as much as possible? *(Checker movie?)*

4.6 Test case #5: Verify weather model

Is the weather downtime model realistic?

Does the weather data represent an average year?

How is the most efficient method for modeling 10 years of weather with minimal rework of simulator code?

How is the number of observations reduced due to weather? *(compare perfect and weather sets)*

4.7 Test case #6: Verify uniformity of observing coverage

For a given science program, are the fields covered uniformly? Note that these results will vary based on proposal characteristics.

Number of fields per filter as a measure of uniformity.

- Plot the distribution of the number of fields observed as a function of filter and time, overplotting the target number:

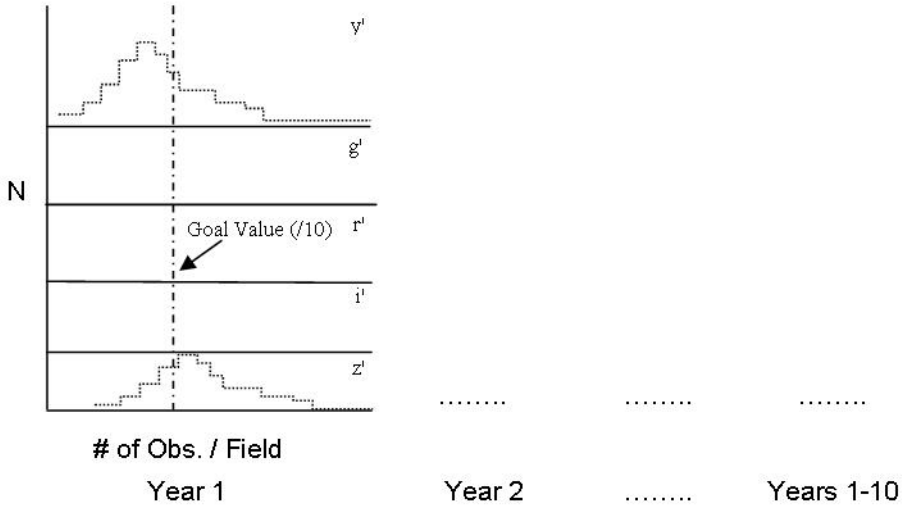

Figure 9.

- What is the number of fields that meet the target per proposal, per filter (need to define narrow range of "meet")?
- What is the number of fields that exceed their target? (Integrate distribution from the goal value out.)
- What fraction of the fields come within 90% of the goal in each filter? (Integrate the distribution appropriately.)
- What is the evolution of this fraction over the 10 years (in one year increments). *(The output of this calculation could be summarized in a table.)*
- Did we observe all the fields? (should be no zeros in the distribution)

Uniformity in spatial coverage by filter.

- What does the spatial sky coverage look like in each filter? Plot the fields observed on a sky projection plot in each filter
- What is the fraction of available sky covered? Compute the fraction of *available* (site dependent) sky that was observed.
- What is the area of sky covered, per year and cumulatively for 10 years, per filter?

- Why is the distribution of number of fields per filter different for different sites (Michelle's grid) – this may not be unexpected as the seeing distribution is different for the different sites.

Seeing as a measure of the uniformity of image quality. What is the range of seeing for a "stack" of images over the sky?

- For all fields, calculate the median, mode, minimum and maximum of the "stack" of observations obtained over 10 years. Plot the distributions of these quantities.
- Make a sky map, color-tag mode (or median) of the seeing in a stack to visually show areas of sky having a most frequent value for seeing in the ranges < 0.6, 0.6-0.8, 0.8-1.0, and >1.0.

Uniformity of depth (limiting magnitude or snr?). *(how do we compute a limiting magnitude?)*

- Does the limiting magnitude (or number of exposures per field) reached after 10 years achieve the science goals?
- How does the limiting magnitude (or number of exposures) per field evolve over the ten years?
- If two filters are specified to the same depth, do we get the right number of exposures? (Check against the ETCs)
- What is the limiting magnitude as a function of lunar cycle and seeing? (the seeing should be a preference not a hard limit)
- For a fixed FOV of 3.5 degrees, what fraction of the observed sky is in the "overlap". Close-ups of a sky map can be plotted to verify this visually.

Uniformity in time.

- What is the number of fields which have a varying timescale for observation?
    - For each series of observations on a field ("stack"), compute its power spectrum.
    - Compute the average and variance for all fields.

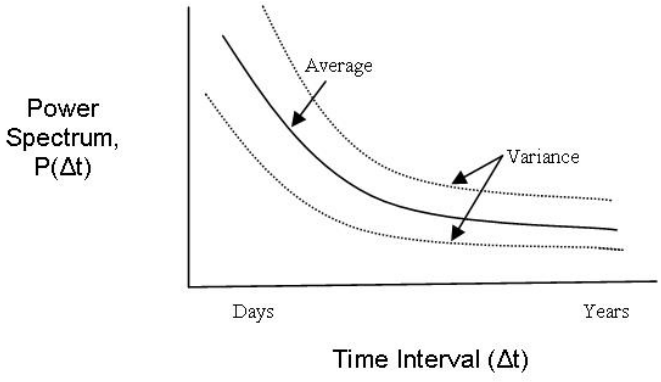

Figure 10.

- How many "serendipitous" series do we have compared to the number of "planned" or programmed series?

4.8 Test case #7: Verify science capabilities

Which proposal dominates the decision making the most?

Proposal specific tests

- On a year average or so, do all science programs get the same number of observations and/or attention?
- Weak Lensing Proposal – is the distribution of observed fields uniform spatially?