SignalSim Design Description

Version 1.5

Author: Jun Mo

2025/1/8

Preface

To be translated.

Copyright

Update History

Version	Date	Description
1.0	2020.7.8	Initial version
1.1	2021.6.29	Add description of XML interpretation and raw measurement generation
1.2	2023.1.21	Add multi-frequency support
1.3	2023.8.26	Add description of API
1.4	2023.11.13	Add support on RINEX4 format
1.5	2025.1.8	Add description of JSON interpretation

1. Design of SignalSim

In the design and testing of GNSS receivers, it is necessary to construct various scenarios, including extreme cases. Since testing all scenarios under real-world conditions is impractical, diverse simulation methods are required to emulate different situations.

The most common simulation tool is the satellite signal simulator. However, satellite signal simulators have several inherent limitations. First, their high cost makes it unrealistic to equip every developer or tester with a dedicated simulator. Second, simulators can only test complete receivers and unable to evaluate individual modules independently. Third, simulators typically provide only trajectory and raw measurement data as reference standards, lacking additional intermediate reference outputs for debugging. Fourth, simulators operate strictly in real time, making them unsuitable for scenarios requiring slowed execution (e.g., testing prototype baseband receiver at lower frequency) or accelerated timelines (e.g., simulating lengthy test scenarios). Similar limitations apply to satellite signal recording and playback devices.

To address these shortcomings, SignalSim—a cost-effective, software-based or hybrid software-hardware simulation platform—has been developed. Designed to output intermediate reference data, SignalSim supports the development and testing of positioning algorithms, baseband tracking programs and hardware baseband signal processing units.

With appropriate hardware acceleration and expansion, it can even function as a simplified satellite signal simulator.

The approach of SignalSim's design is to build a platform that reconstructs satellite signals through following steps:

- ✓ Transmission Time Calculation: Determines the precise transmission time of each satellite signal at a specific receiver moment.
- ✓ Baseband Signal Generation: Generates baseband signals based on the computed transmission times.
- ✓ Interpolation & Sampling: Produces digital intermediate frequency (IF) sampled signals by interpolating baseband signals over defined time intervals.
- ✓ Upconversion: Converts IF signals to radio frequency (RF) signals.

Each step outputs intermediate results, enabling developers to validate and debug corresponding receiver modules (e.g., baseband processing, positioning algorithms) at critical stages of the signal chain. This modular workflow ensures flexibility and transparency throughout the receiver development lifecycle.

The core part of SignalSim is to generate the intermediate frequency signal at a specific time. The formula used to generate the IF signal is:

$$s_{IF}(t) = \sum_{n=1}^{N}\{\sqrt{2P_n}D_n(t - T_p - \delta t_{iono})C_n(t - T_p - \delta t_{iono})\cos[\omega_{IF}t - \omega_{RF}(T_p - \delta t_{iono} + \phi_0)]\} + noise$$

In above equation, P_n is the power of satellite signal, C_n and D_n are spreading code and modulation data respectively. ω_{IF} and ω_{RF} are IF and RF frequency. T_p is signal propagation time with correction of satellite clock error, earth rotate, theory of relativity and troposphere delay. δt_{iono} is ionosphere delay. ϕ_0 is initial carrier phase which can be simply set to 0.

It should be noted that for each specific GNSS receiver, ϕ_0, ω_{IF} and ω_{RF} are constants, and T_p and δt_{iono} changes with time. Signal power P_n may also changes in different environment and different elevation. All variables above are slow-changing variables except for T_p.

With the definition of code phase $T_c = t - T_p - \delta t_{iono}$ and carrier phase $\Phi = 1/2\pi\,[\omega_{IF}t - \omega_{RF}(T_p - \delta t_{iono})]$, the equation of IF signal can be rewrite as：

$$s_{IF}(t) = \sum_{n=1}^{N}\{\sqrt{2P_n}D_n(T_c)C_n(T_c)\cos[2\pi\Phi]\} + noise$$

In the above equation, the unit of code phase is second and the unit of carrier phase is cycle. With such definition, the integer part of carrier phase can be omitted.

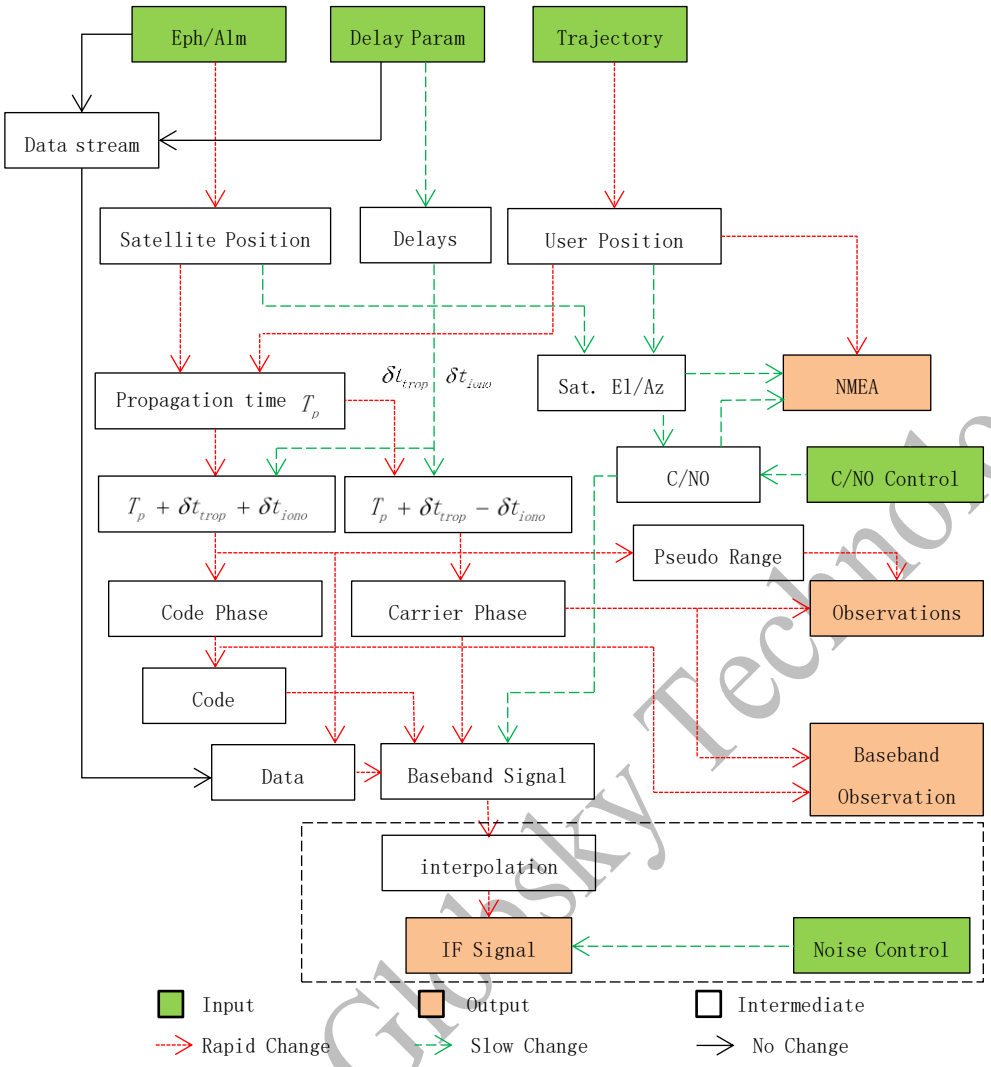
For complex value signal, we have similar equation:

$$s_{IF}(t) = \sum_{n=1}^{N}\{\sqrt{2P_n}B_n e^{i[\omega_{IF}t - \omega_{RF}(T_p - \delta t_{iono} + \phi_0)]}\} + noise$$

For each satellite

$$B = D_I(t - T_p - \delta t_{iono})C_I(t - T_p - \delta t_{iono}) + iD_Q(t - T_p - \delta t_{iono})C_Q(t - T_p - \delta t_{iono})$$

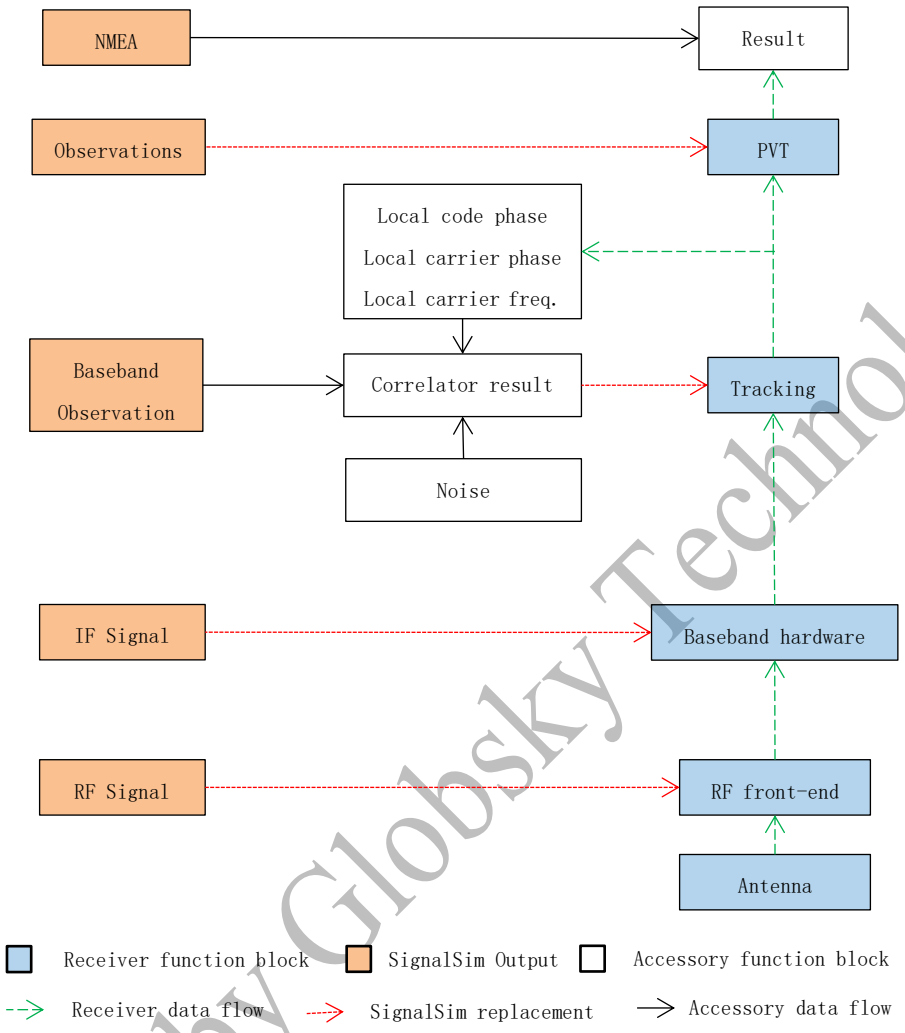
The block diagram of SignalSim is shown below:



In the above figure, the green blocks represent the input control including receiver trajectory, ephemeris/almanac, model parameters of ionosphere/troposphere delay, signal power control and noise model control.

The complete SignalSim can include outputs from four distinct stages. The first is the output of receiver positions that can serve as positioning reference results, which may be in NMEA format or other formats. The second is the output of raw observations, which can be used for debugging positioning algorithms. The third is baseband observations, including code phase and carrier phase of satellite signals, which can be utilized for debugging baseband tracking programs. The fourth is digital intermediate frequency (IF) sampling, which can be employed to debug baseband signal processing logic. If the components within the dashed box adopt hardware-accelerated real-time output, they can further undergo up-conversion to generate radio frequency (RF) signals.

The figure below illustrates the correspondence between these outputs and the receiver's processing modules. It demonstrates that SignalSim implements a reverse receiver processing flow.



On the development of each GNSS receiver functional blocks, SignalSim is able to provide replacement as input data, which helps easy debug method.

The organization of the control parameters uses JSON format for easy editing and expansion. Please refer “SignalSim JSON format specification” for details of the JSON file format.

In addition to the above basic functions, SignalSim can also be expanded as needed, such as adding data such as inertial sensors. The data source of the inertial sensor can be generated based on the trajectory file and the carrier posture, and the sensor error model can be added. In this way, loosely coupled/tightly coupled, or even deeply coupled algorithm research can be carried out.

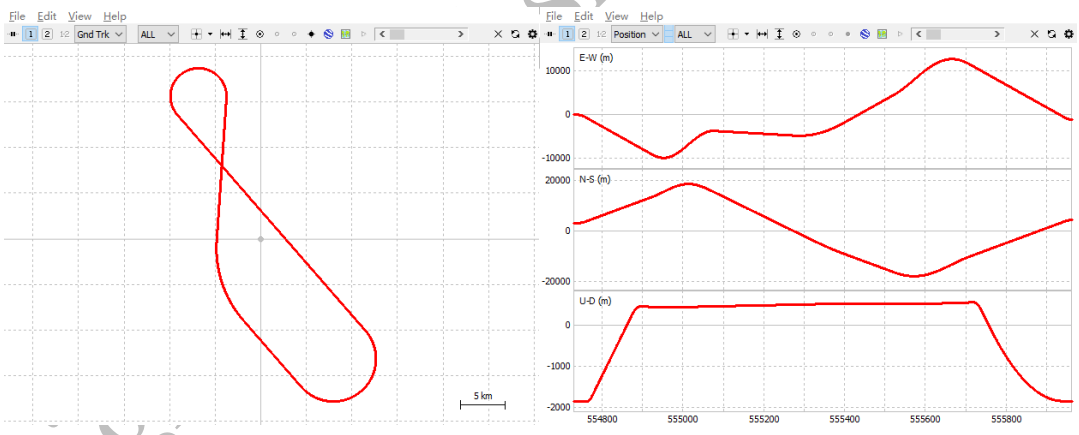
2. Receiver Trajectory

Receiver trajectory defines the kinematic movement of the receiver as an important part of input parameters of SignalSim. In order to make the receiver trajectory design more flexible and calculate the points on the trajectory more accurately, the receiver trajectory is designed by trajectory splicing instead of feature points.

The trajectory first starts with the initial position and velocity of the receiver, and then based on the starting point, the entire trajectory is spliced by connecting different types of trajectories segment by segment. Each segment of trajectories are selected from one of the following types:

- ✓ Uniform linear motion
- ✓ Uniformly accelerated linear motion
- ✓ Vertically accelerated motion
- ✓ Variable accelerated linear motion
- ✓ Horizontal turn

The trajectory of most vehicles can be simulated by splicing the above-mentioned trajectories. For example, the following figure shows a flight trajectory.



The flight path first moves toward the northwest, then accelerates to take off. After reaching a certain altitude, it turns 225 degrees clockwise, then turns 45 degrees counterclockwise after continuing to fly, and then turns 180 degrees counterclockwise after flying for a while, and finally descends to return to the original position.

This kind of trajectory splicing method only needs to change the initial position and speed, and the same trajectory can be placed at any desired position with different starting positions and angles. Of course, since the earth is an ellipsoid, for a relatively large range of motion trajectories, the original horizontal movement is no longer horizontal after a certain distance. This is because the coordinate system of the trajectory is Euclidean space. Although the conversion relationship between the

position and velocity in the ECEF coordinate system and the position and velocity in the local coordinate system will be recalculated according to the starting position of the trajectory during the actual calculation of each segment of the trajectory, due to the different conversion matrices at each position, the final trajectory may have a certain deviation from the spherical coordinate system. This deviation is not caused by the cumulative calculation error, but by the difference between the rectangular coordinates and the spherical coordinates. Therefore, the reference trajectory generated based on this is still consistent with the original observation and will not affect the accuracy of the simulation platform output. In addition, if the vehicle's motion range is not too large, this error can be ignored, and if the vehicle's motion exceeds a certain range, it is recommended to make appropriate adjustments. For example, if the horizontal straight flight reaches more than tens of kilometers, at this time, according to the straight line, the altitude will gradually increase, and the vehicle will have a vertical component on velocity. At this time, a vertical acceleration motion can be added for adjustment, and the vertical velocity decrease to 0 or a negative value. This is equivalent to fitting a circle with multiple straight lines.

3. Raw Measurement Generation

Generally, the raw measurements referred here include pseudo-range, carrier phase (or equivalently ADR), instant Doppler and C/N0 for each satellite. In a real GNSS receiver, the calculation of signal code phase and carrier phase uses the local replica. Within SignalSim, the raw measurements are calculated based on known receiver and satellite position.

The equation to calculate pseudo-range is $\rho = c(T_r - T_t)$. When the local clock error is ignored, the signal transmitted at T_c will be received at time $t = T_c + T_p + \delta t_{iono}$ in which T_p represents propagation time (including all corrections except ionosphere delay). So we get $\rho = cT_p + c\delta t_{iono}$.

The carrier phase (ADR) includes integer part and fractional part. Similar to pseudo-range, the phase of carrier can be calculated use:

$$\Phi = \frac{1}{2\pi}[\omega_{IF}t - \omega_{RF}(T_p - \delta t_{iono})] = f_{IF}t - f_{RF}(T_p - \delta t_{iono})$$

The nominal IF is usually 0 or multiple of kHz, and the local observation epoch is also usually defined to be at second or millisecond boundary. This makes $f_{IF}t$ be an integer value. The ADR observation has flexible integer part and fractional part has negative value to the phase of carrier, so we define ADR as:

$$\hat{\Phi} = -\Phi + N = -f_{IF}t + f_{RF}(T_p - \delta t_{iono}) + N = -f_{IF}(t - t_0) + f_{RF}(T_p - \delta t_{iono})$$

Here $N = f_{IF}t_0$ is an artificially defined integer value at initial epoch t_0.

So when $t = t_0$, $\hat{\Phi} = f_{RF}(T_p - \delta t_{iono}) = f_{RF}T_p - \frac{1}{\lambda}c\delta t_{iono}$

At another epoch $t = t_1$, the first term $-f_{IF}\Delta t = -f_{IF}(t_1 - t_0)$ equals whole cycle part introduced in Φ due to non-zero IF and should be removed. So we still get $\hat{\Phi} = f_{RF}T_p - \frac{1}{\lambda}c\delta t_{iono}$. Also an integer number $\hat{N}$ can be added to ADR to result equation $\hat{\Phi} = f_{RF}T_p - \frac{1}{\lambda}c\delta t_{iono} + \hat{N}$. Different satellite can have different $\hat{N}$. For the same satellite, the value of $\hat{N}$ will be the same between epochs. Half integer $\hat{N}$ can be applied to simulate half cycle ambiguity.

Instant Doppler can be calculated by first calculate the relative velocity between satellite and receiver: $v = (\vec{v_s} - \vec{v_r}) \cdot \hat{\vec{r}}$. Here $\vec{v_s}$ is the velocity of satellite, $\vec{v_r}$ is the velocity of receiver and $\hat{\vec{r}}$ is the unit vector from receiver to satellite. The instant Doppler is $f_D = -v/\lambda = \frac{1}{\lambda}(\vec{v_s} - \vec{v_r}) \cdot \hat{\vec{r}}$. The satellite clock error can be ignored if it

is very small. Otherwise a correction term needs to be added: $f_D = -v/\lambda = \frac{1}{\lambda}(\overrightarrow{v_s} - \overrightarrow{v_r}) \cdot \hat{\vec{r}} + f_{RF}[af_1 + af_2(t - t_{OC})]$

or equivalently

$f_D = -v/\lambda = \frac{1}{\lambda}\{(\overrightarrow{v_s} - \overrightarrow{v_r}) \cdot \hat{\vec{r}} + c[af_1 + af_2(t - t_{OC})]\}$

The af_2 term can be ignored in most cases (generally $af_2 < 10^{-14}$ and $t - t_{OC} < 10^4$, so the corresponding term will be less than 0.01Hz).

CN0 can be calculated using pre-defined satellite signal, elevation angle and other user defined signal fading methods.

4. Generation of correlation result

The development and debugging the program to do baseband signal tracking is usually frustrated because it is usually hard to reproduce the problem and once you stopped at a break point, the real signal will not wait for you to continue the tracking process. There is no satisfied solution until correlator behavior model is adopted with the support of SignalSim.

In most ASIC/FPGA based receiver, the hardware will do signal correlation as a digital signal processing acceleration unit. So SignalSim will help to generate correlation result. In most GNSS receiver, the correlation period is 1 millisecond (or multiple of 1ms if additional coherent summation is adopted). The method to generate 1ms correlation result will be described in the following paragraphs. With the correlation result, engineers can further create behavior model of hardware tracking engine according to customer defined interface.

Besides the satellite code, carrier phase and signal power, SignalSim also need local code phase and carrier phase to generate correlation result. The equation used to calculation correlation result is:

$$Cor_n = A \cdot R(|\Delta t + (n-p)\Delta t_I|) \cdot Sinc(\Delta \bar{f} \cdot \mathrm{T}) \cdot e^{j\Delta\bar{\phi}} + \sigma N$$

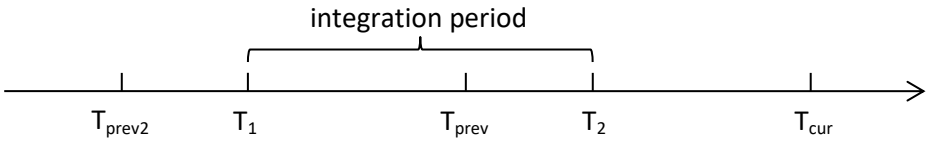
Here n represents the index of correlator, A is the amplitude of the signal which depend on the signal power. Δt is the signal-local code phase difference, Δt_I is the interval between each correlator, p is the index of peak correlator. $\Delta \bar{f}$ is the average signal-local frequency difference during integration period. T is integration period which is usually 1ms. $\Delta\bar{\phi}$ is the average signal-local phase difference during integration period. σ is the standard deviation of noise, N is Complex white noise with standard Gaussian distribution. R() is either standard correlation function defined as following:

$$R(\Delta t) = \begin{cases} 1 - \dfrac{\Delta t}{T_c}, \Delta t \le T_c \\ 0, \Delta t > T_c \end{cases}$$

Or can be adjusted according to signal bandwidth.

The average frequency and phase difference can be calculated using following method:

The code phase difference between signal and local replica will not change significantly, so it can be treated as a constant value during integration period. The average carrier frequency and phase difference need to be calculated using linear interpolation method as following:



In the above figure, T_{cur} is current observation epoch, T_{prev} and T_{prev2} are previous and the one before previous observation epoch respectively. T_1 to T_2 is correlation period (normally 1ms) align to code period. The correlation value is generally accessible by CPU at epoch T_{cur}.

Define $\alpha = \frac{T_2 - T_{prev}}{T_{cur} - T_{prev}}$ as the ratio of integration period in the most recent millisecond. Equivalently, the code phase at current epoch divided by code period is $1 - \alpha$.

If the signal frequency is considered to be linear change and local frequency is constant value between each observation epochs, we will get average signal Doppler in time segment of T_1 to T_{prev} and T_{prev} to T_2 are:

$$f_{D1} = f_{D,prev2} + \frac{1+\alpha}{2}\left(f_{D,prev} - f_{D,prev2}\right) = \frac{1-\alpha}{2} f_{D,prev2} + \frac{1+\alpha}{2} f_{D,prev}$$

$$f_{D2} = f_{D,prev} + \frac{\alpha}{2}\left(f_{D,cur} - f_{D,prev}\right) = \frac{2-\alpha}{2} f_{D,prev} + \frac{\alpha}{2} f_{D,cur}$$

In which $f_{D,cur}$, $f_{D,prev}$ and $f_{D,prev2}$ are signal Doppler at corresponding epoch. Then the average frequency difference during T_1 to T_2 is: The local frequency at epoch，如果在 Tprev2 和 Tprev 两个时刻，跟踪环路设置的本地载波频率分别是 fLO,prev2 和 fLO,prev，则在 T1 到 T2 的时间段内，平均的频率差为：

$$\Delta\bar{f} = (1-\alpha)\left(f_{D1} - f_{LO,prev2}\right) + \alpha\left(f_{D2} - f_{LO,prev}\right)$$

In which $f_{LO,prev}$ and $f_{LO,prev2}$ are local frequency at corresponding epoch.

Similarly, with the signal carrier phase at three epochs are $\Phi_{s,prev2}$, $\Phi_{s,prev}$ and $\Phi_{s,cur}$ respectively, the local carrier phase are $\Phi_{LO,prev2}$, $\Phi_{LO,prev}$ and $\Phi_{LO,cur}$, the average phase difference is:

$$\Delta\bar{\phi} = (1-\alpha)\left(\frac{1-\alpha}{2}\Delta\phi_{prev2} + \frac{1+\alpha}{2}\Delta\phi_{prev}\right) + \alpha\left(\frac{2-\alpha}{2}\Delta\phi_{prev} + \frac{\alpha}{2}\Delta\phi_{cur}\right)$$

with $\Delta\phi = 2\pi(\Phi_s - \Phi_{LO})$ at the unit of radian.

The value of $\Delta\bar{f}$ and $\Delta\bar{\phi}$ are the same for different delay correlators while correlation result are different when different delay applied to correlation function R().

It should be noted that the noise between each correlator is correlated, because the local codes between adjacent correlators are partially overlapped. Therefore, when generating the noise of each correlator, it must also be taken into account. Since the noise is a complex number, the correlation matrix of the real or imaginary part of the noise between each correlator is:

$$\mathbf{R} = \begin{bmatrix} 1 & 1-\Delta & 1-2\Delta & \cdots & 0 \\ 1-\Delta & 1 & 1-\Delta & \cdots & 0 \\ 1-2\Delta & 1-\Delta & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

That is, the main diagonal element is 1, indicating that the autocorrelation is 1; the secondary diagonal element is $1-\Delta$ indicating that the correlation coefficient of the noise between adjacent correlators is $1-\Delta$; the third diagonal element is $1-2\Delta$ indicating that the correlation coefficient of the noise between the correlators with an interval of 2 is $1-2\Delta$ and so on. Here Δ is the ratio of adjacent correlator interval and length of code chip. If the correlator interval is 1/2 chip, then $\Delta = \frac{1}{2}$; if correlator interval is 1/8 chip, then $\Delta = \frac{1}{8}$.

Since the correlation matrix **R** is a symmetric positive definite matrix, the above matrix can be applied by Cholesky decomposed to obtain a triangular matrix **C** while $\mathrm{R} = \mathrm{C} \cdot \mathrm{C}^{\mathrm{T}}$. When N correlated complex noises need to be generated, N pairs of independent and identically distributed Gaussian white noises are first generated to form N complex noises, and then the triangular matrix **C** is multiplied to the independent N complex noises to obtain N correlated noises that meet the correlation relationship **R**.

5. IF Signal Generation

By calculating the code phase and carrier phase at a specific moment, according to the formula in Chapter 1, the intermediate frequency signal at each sampling moment can be calculated. However, in order to improve the calculation speed, the code phase and carrier phase at all sampling moments will not be calculated due to huge amount of calculation effort.

Instead, the code phase and carrier phase will be calculated at a certain time interval, and the sampling time within the time interval will be calculated using linear interpolation. If the dynamics are relatively large, the time interval can be 1ms, and if the dynamics are relatively small, the time interval can be 10ms.

6. SignalSim API Description

The commonly used functions provided by the SignalSim software package are divided into several categories: functions to process scenario configuration file, functions to calculate receiver trajectory, functions to generate raw measurements, functions to read and synthesis navigation messages, and auxiliary functions, etc.

6.1 Scenario Configuration

The process of JSON file is implemented by class CJsonStream. The most commonly used member functions are:

- `ReadFile(): Input parameter as file name, read and interpret the contents then stored in a tree-organized CJsonObject structure.`
- `GetRootObject(): Return the root node of the CJsonObject tree.`
- `GetFirstObject(): Return the pointer of the content object or the first element of the link-list with given CJsonObject.`
- `GetNextObject(): Return pointer to the next content in the link-list.`

The last two functions are used to access the entire contents of the loaded JSON file traversal.

The read and interpret of XML file is similar. The functions will not be described here because XML format is obsolete.

6.2 Receiver Trajectory

The receiver trajectory generation is implemented by class CTrajectory, which organize a list of CTrajectorySegment. The class CTrajectorySegment is the base class for classes deal with different trajectory type. The member functions provided by CTrajectory class are:

- `SetInitPosVel(): Set the initial position and velocity.`
- `AppendTrajectory(): Add a new trajectory segment`
- `ResetTrajectoryTime(): Reset the time to the beginning of trajectory`
- `GetNextPosVelECEF(): Get the receiver position and velocity of next epoch with given time interval between epochs.`

6.3 Raw measurements

A structure SATELLITE_PARAM is used to store all parameters used to calculate raw measurements. At a given epoch, member variables of the above structure for each satellite is first calculated, then the contents are used to further calculate the raw

measurements (also correlation results and IF signal). The commonly used functions are listed below:

- `GetSatelliteParam()`: Calculate satellite parameters at a given epoch.
- `GetSatelliteCN0()`: Assign CN0 element in the structure.
- `GetTravelTime()`: Calculate propagation time for specific signal.
- `GetCarrierPhase()`: Get carrier phase.
- `GetDoppler()`: Get instant Doppler.

The return value of the latter three functions will be different for different signals broadcasted by the same satellite at the same time, so the signal ID is used as input parameter.

6.4 Navigation Data Read and Compose

The navigation data is first read from navigation file. The most commonly used navigation file format is RINEX .n file. SignalSim also support reading YUMA format and customer defined almanac. If there is no almanac file, SignalSim is able to generate almanac from ephemeris.

The navigation data read from RINEX file is implemented in CNavData class, which provide following member functions:

- `ReadNavFile()`: Read RINEX format navigation data including ephemeris, ionosphere parameters, UTC parameters, system time difference between different constellations etc. The ephemeris will be stored in a ephemeris pool.
- `ReadAlmFile()`: Read almanac from file. File format current supported is YUMA for GPS and XML for Galileo. BDS and GLONASS need conversion by user.
- `CompleteAlmanac()`: Complete the missing almanac. If part of the almanac of one constellation is missing, ephemeris of corresponding satellite will be used to convert to almanac at reference time of other valid almanac. If no valid almanac is available, given reference time is adopted.
- `FindEphemeris()`: Find the valid ephemeris for specified satellite at given time from the ephemeris pool and return the pointer. If no valid ephemeris found, a NULL pointer will be returned.
- `FindGloEphemeris()`: Same as FindEphemeris except this function is specified for GLONASS satellite.
- `GetGlonassSlotFreq()`: Get frequency ID of a GLONASS satellite with given slot number.

- `GetGpsIono(), GetBdsIono(), GetGalileoIono(): Return ionosphere parameters of corresponding constellation.`
- `GetGpsUtcParam(), GetBdsUtcParam(), GetGalileoUtcParam():Return UTC parameters of corresponding constellation.`

Classes to do navigation data composition is used to generate broadcasting bit streams of a satellite. All classed derived from the same base class NavBit and provide following member functions:

- `SetEphemeris(): Set ephemerides for corresponding message type. For GNAV message, the first parameter is slot number and the type of second parameter need to be casted to PGLONASS_EPHEMERIS format.`
- `SetAlmanac(): Set almanacs for corresponding message with parameter of almanac array.`
- `SetIonoUtc(): Set ionosphere and UTC parameters.`
- `GetFrameData(): Obtain bit stream of a subframe/page at given time. The bit stream is a binary sequence stored in a array allocated in caller. The length of the array depends on the type of the message. The function will assign the stream of complete subframe/page containing given time epoch. The following table gives the length of the array for each type of message:`

Message Type	Array Length	Note
LNAV	300	Length of one subframe
CNAV	600	
CNAV2	1800	
D1/D2	300	
B-CNAV1	1800	
B-CNAV2	600	
B-CNAV3	1000	
I/NAV	250	Length of one page
F/NAV	500	
GNAV	200	One string after encoded with meander code

6.5 Satellite Signal Generation

The class CSatelliteSignal is used to generate bare satellite signal. Bare satellite signal means the signal is not modulated by spread code or carrier, only data stream (and secondary code modulation on pilot channel) is applied. The amplitude of the signal is normalized to 1, which means data and pilot channel share the total power of 1. The generation of satellite signal can be unified to the same interface while difference

signal has different components and different data/pilot phases. Before getting the bare satellite signal, function SetSignalAttribute() need to be called first to specify satellite constellation, signal ID, satellite ID and navigation data.

Because the data signal and pilot signal may have different phase, the signals returned are complex value. Generally, if the corresponding signal only has data component, then the phase of the data signal is either 0 or π when modulated with 0 or 1. If the signal has both data and pilot component, the phase of the pilot signal is either 0 or π when modulated with 0 or 1; and the phase of data signal depends on the phase relationship between data and pilot

6.6 Other functions

Other auxiliary functions including satellite position/velocity calculation using ephemeris, time conversion functions, functions for RINEX file read and interpretation etc.

7. Support for RINEX4 Format

The RINEX4 format extended the support on different navigation data adopted by modern signal. For GPS, CNAV data on L5 and CNAV2 data on L1C are added; for BDS, CNAV1/2/3 data at frequency B1C, B2a and B2b are added; Galileo add the distinction flag for I/NAV and F/NAV.

The navigation data adopted by modern signals has similar satellite orbit and clock parameters except the following changes: A_dot and delta_n_dot parameters are added to orbit parameters; besides TGD, ISC parameters as inter-signal correction for all signals are added; definitions of the flags (especially health and accuracy flags) are changed.

The supports for RINEX4 format in SignalSim are described here.

7.1 Read of RINEX File

SignalSim will automatically determine whether the data segment is the original RINEX 3 format message data or the data segment starting with the RINEX 4 format data header when reading the navigation message file. Then it will determine the navigation message type and assign a value based on the data read in. For the sake of uniformity, except for the navigation messages of GLONASS and SBAS, all other ephemeris data use the same ephemeris data structure. Because the message type of QZSS is the same as that of GPS, so they are treated as the same message. The arrangement of ephemeris data of different formats in the RINEX navigation message file is as follows:

		GPS/QZSS			Galileo	BDS				NavIC
line		LNAV	CNAV	CNV2	I/FNAV	D1/D2	CNAV1	CNAV2	CNAV3	LNAV
0	Time	t_{oc}	t_{oc}	t_{oc}	t_{oc}	t_{oc}	t_{oc}	t_{oc}	t_{oc}	t_{oc}
	0	af0	af0	af0	af0	af0	af0	af0	af0	af0
	1	af1	af1	af1	af1	af1	af1	af1	af1	af1
	2	af2	af2	af2	af2	af2	af2	af2	af2	af2
1	3	IODE	A_{dot}	A_{dot}	IOD_{nav}	AODE	A_{dot}	A_{dot}	A_{dot}	IODEC
	4	C_{rs}	C_{rs}	C_{rs}	C_{rs}	C_{rs}	C_{rs}	C_{rs}	C_{rs}	C_{rs}
	5	Δn	Δn0	Δn0	Δn	Δn	Δn0	Δn0	Δn0	Δn
	6	M0	M0	M0	M0	M0	M0	M0	M0	M0
2	7	C_{uc}	C_{uc}	C_{uc}	C_{uc}	C_{uc}	C_{uc}	C_{uc}	C_{uc}	C_{uc}
	8	e	e	e	e	e	e	e	e	e
	9	C_{us}	C_{us}	C_{us}	C_{us}	C_{us}	C_{us}	C_{us}	C_{us}	C_{us}
	10	sqrtA	sqrtA	sqrtA	sqrtA	sqrtA	sqrtA	sqrtA	sqrtA	sqrtA
3	11	t_{oe}	t_{op}	t_{op}	t_{oe}	t_{oe}	t_{oe}	t_{oe}	t_{oe}	t_{oe}
	12	C_{ic}	C_{ic}	C_{ic}	C_{ic}	C_{ic}	C_{ic}	C_{ic}	C_{ic}	C_{ic}
	13	Ω_0	Ω_0	Ω_0	Ω_0	Ω_0	Ω_0	Ω_0	Ω_0	Ω_0
	14	C_{is}	C_{is}	C_{is}	C_{is}	C_{is}	C_{is}	C_{is}	C_{is}	C_{is}
4	15	i_0	i_0	i_0	i_0	i_0	i_0	i_0	i_0	i_0
	16	C_{rc}	C_{rc}	C_{rc}	C_{rc}	C_{rc}	C_{rc}	C_{rc}	C_{rc}	C_{rc}
	17	ω	ω	ω	ω	ω	ω	ω	ω	ω
	18	Ω_{dot}	Ω_{dot}	Ω_{dot}	Ω_{dot}	Ω_{dot}	Ω_{dot}	Ω_{dot}	Ω_{dot}	Ω_{dot}
5	19	i_{dot}	i_{dot}	i_{dot}	i_{dot}	i_{dot}	i_{dot}	i_{dot}	i_{dot}	i_{dot}
	20	CodeL2	$\Delta n0_{dot}$	$\Delta n0_{dot}$	source		$\Delta n0_{dot}$	$\Delta n0_{dot}$	$\Delta n0_{dot}$	
	21	Wn	URA_{NED0}	URA_{NED0}	Wn	Wn	SatType	SatType	SatType	Wn
	22	L2P	URA_{NED1}	URA_{NED1}			t_{op}	t_{op}	t_{op}	
6	23	Accuracy	URA_{ED}	URA_{ED}	SISA	Accuracy	$SISAI_{oe}$	$SISAI_{oe}$	$SISAI_{oe}$	URA
	24	health	health	health	health	health	$SISAI_{ocb}$	$SISAI_{ocb}$	$SISAI_{ocb}$	health
	25	T_{GD}	T_{GD}	T_{GD}	B_{GDE5a}	T_{GD1}	$SISAI_{oc1}$	$SISAI_{oc1}$	$SISAI_{oc1}$	T_{GD}
	26	IODC	URA_{NED2}	URA_{NED2}	B_{GDE5b}	T_{GD2}	$SISAI_{oc2}$	$SISAI_{oc2}$	$SISAI_{oc2}$	
7	27	T_{tr}	ISC_{L1CA}	ISC_{L1CA}	T_{tr}	T_{tr}	ISC_{B1Cd}		SISMAI	T_{tr}
	28	fit interval	ISC_{L2C}	ISC_{L2C}		AODC		ISC_{B2ad}	health	
	29		ISC_{L5I}	ISC_{L5I}			TGD_{B1Cp}	TGD_{B1Cp}	B2b flag	
	30		ISC_{L5Q}	ISC_{L5Q}			TGD_{B2ap}	TGD_{B2ap}	T_{GDB2b}	
8	31		T_{tr}	ISC_{L1Cd}			SISMAI	SISMAI	T_{tr}	
	32		Wn_{op}	ISC_{L1Cp}			health	health		
	33						B1C flag	Integ. Flag		
	34						IODC	IODC		
9	35			T_{tr}			T_{tr}	T_{tr}		
	36			Wn_{op}						
	37									
	38						IODE	IODE		

Same meaning
Similar meaning
Bit field

The parameter A_dot and delta_n_dot are added. These two parameters will be assigned to 0 for the messages without them. The zero value will not affect the result when calculating the satellite position and velocity.

The ISC parameters will be converted to a tgd array. According to the satellite clock error calculation methods provided in ICDs, a uniformed equation $(\Delta t_{SV})_{(i)} = \Delta t_{SV} - T_{GD(i)}$ is used to calculate clock error for different signal. For backward compatibility, tgd and tgd2 fields keep unchanged while a new array tgd_ext[5] is added. The definition of tgd_ext[] will be described later.

The flag and health fields are extended from 8bits to 16bits and has the following definitions for different constellation:

flag																
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPS													fit	L2P	Code L2	
Galileo							TOC ref							source		
BDS						B2b integ. flags			B2a integ. flags			B1C integ. flags			SatType	

health																
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPS							L1C	L1	L2	L5	LNAV health					
Galileo								E5b HS/DVS			E5a HS/DVS			E1B HS/DVS		

7.2 Conversion Between Different Message Types

The user of SignalSim may need to generate signals with some of the fields in navigation data is missing. For example, with only BDS D1/D2 navigation data, user may need to generate B1C signal which requires CNAV1 message. Another example is to generate GPS signal on all frequencies with only RINEX3 data provided. In such cases, completion on missing fields is needed.

The parameter A_dot and delta_n_dot can be assigned to 0 when they are unavailable. And the flag/health fields can be applied from one frequency to other frequencies. The T_{GD} fields need special attention. The following table shows how the corresponding fields are assigned:

			tgd_ext				
	tgd	tgd2	0	1	2	3	4
	L1CA B1I E1	L2C B2I	L1Cd B1Cd	L1Cp B1Cp	L5I B2ad E5a	L5Q B2ap	B2bI E5b
LNAV	T_{GD}	γT_{GD}	T_{GD}	T_{GD}	$\gamma_{L5} T_{GD}$	$\gamma_{L5} T_{GD}$	T_{GD}
CNAV	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	T_{GD}
CNAV2	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	$T_{GD} - \mathrm{ISC}_i$	T_{GD}
D1/D2	T_{GD1}	T_{GD2}	T_{GD1}	T_{GD1}	$\gamma_{L5} T_{GD1}$	$\gamma_{L5} T_{GD1}$	$\gamma_{B2b} T_{GD1}$
CNAV1	TGD_{B1C}	$\gamma_{B2I} TGD_{B1C}$	$TGD_{B1C} - \mathrm{ISC}$	TGD_{B1C}	TGD_{B2a}	TGD_{B2a}	$\gamma_{B2b} TGD_{B1}$
CNAV2	TGD_{B1C}	$\gamma_{B2I} TGD_{B1C}$	TGD_{B1C}	TGD_{B1C}	$TGD_{B2a} - \mathrm{ISC}$	TGD_{B2a}	$\gamma_{B2b} TGD_{B1}$
CNAV3	TGD_{B2bI}/γ_{B2b}	TGD_{B2bI}	TGD_{B2bI}/γ_{B2b}	TGD_{B2bI}/γ_{B2b}	TGD_{B2bI}	TGD_{B2bI}	TGD_{B2bI}
I/NAV	$BGD_{(E1,E5a)}$	$BGD_{(E1,E5b)}$	-	-	$\gamma_{E5a} BGD_b$	-	$\gamma_{E5b} BGD_b$
F/NAV	$BGD_{(E1,E5a)}$	$BGD_{(E1,E5a)}$	-	-	$\gamma_{E5a} BGD_a$	-	$\gamma_{E5b} BGD_a$

The left column shows the data source. The green color means that the value can be assigned directly use the data in RINEX file. The blue color means that the value is assigned use TGD/ISC data for other frequencies.

7.3 Selection of the Ephemeris

Since a satellite will broadcast different messages at different frequencies, the user has the flexibility to choose different message for different signal.

The member function FindEphemeris() of the class CNavData class has an additional parameter to indicate the preferred ephemeris source to match the content of the source field in the ephemeris structure. The ephemeris data come from the corresponding frequency will be matched first. If this parameter is not specified, the default value 0 is used, and the corresponding definition is the LNAV/D1D2 ephemeris data in the original RINEX 3 file, which is compatible with the original interface and function.

The following code gives an example of generating corresponding navigation messages for signals at two frequencies, B1I and B1C:

```
    for (i = 1; i <= TOTAL_BDS_SAT; i ++)
    {
        BdsEphemeris = NavData.FindEphemeris(BdsSystem, CurTime, i);
        NavBitArray[0]->SetEphemeris(i, BdsEph[i - 1]);
        BdsEphemeris = NavData.FindEphemeris(BdsSystem, CurTime, i,
EPH_SOURCE_CNV1);
        NavBitArray[1]->SetEphemeris(i, BdsEph[i - 1]);
    }
```

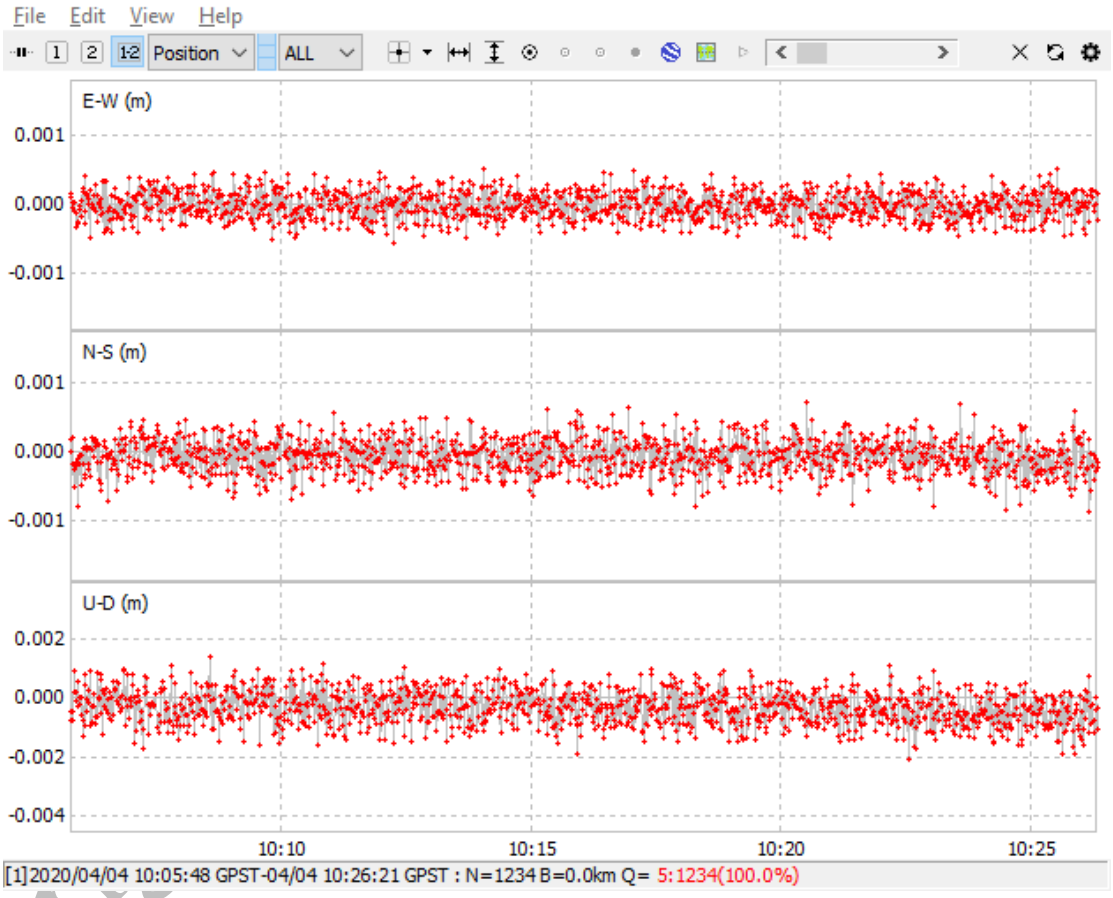
The pointer in `NavBitArray[0]` and `NavBitArray[1]` point to instances with type `D1D2NavBit` and `BCNavBit` respectively. The pointer can be used as parameter when calling function `SetSignalAttribute()`. Then navigation messages from different source can be applied to different signal.

It should be noted that each satellite has only one true position at a given time while ephemeris in different message may give the position with a little difference. The user needs to choose ephemeris from only one source to calculate the satellite position.

Annex A Example of Raw Measurement Generation

The project JsonObsGen gives an example of reading configurations from JSON file and generate RINEX format observation file. The file test_obs2.json is the example of configuration file and the output is test2.o. The configuration file can be modified a little to generate receiver reference position.

By using the toolset in RTKLIB, the observation can be processed through RTKPOST to do PVT calculation. The difference between PVT result and the reference position is shown below using RTKPLOT. The difference comes from the truncation of the raw measurement in RINEX file.



Annex B Ephemeris to Almanac conversion

When the almanac information is missing, the almanacs of a given week number and reference time can be converted from the ephemerids. This is because the accuracy of the ephemeris is higher than that of the almanac, so the correction information can be ignored and the rough approximate orbital parameters can be retained. For the almanac parameter design of most satellite signals, it is relatively simple to convert the almanac from the ephemeris. In this case the almanac is a simplified version of the ephemeris parameters, some second-order corrections are removed, and the accuracy of the parameters is reduced. For this type of almanac conversion, it is only necessary to directly assign values to the parameters that are not related to the reference time, and convert the parameters related to the reference time according to the difference in the reference time between ephemeris and almanac.

There are two types of special almanac conversion methods, one is the GLONASS almanac. Not only because the arrangement of GLONASS almanac parameters different from those commonly used by other satellite signals, but also the ephemeris format of GLONASS satellites is also different from other satellites. The other is the GEO satellite of the BDS system. The coordinate system used by the ephemeris of the GEO satellite is a custom inclined coordinate system, and the conversion relationship between this coordinate system and the ECEF coordinate system is not fixed. Therefore, the coordinate system conversion of orbital parameters becomes very complicated. The above two conversion methods will be introduced separately later.

For the almanac that generally converts a given time from the ephemeris, the parameters that are unrelated to the reference time are first assigned. These parameters include: satellite semi-major axis, eccentricity and orbital inclination, perigee inclination and rate of change of ascending node right ascension. Similarly, the clock drift of the satellite clock parameter clock is also unrelated to the reference time. Next, the time difference dt of each reference time is calculated to correct other parameters. These parameters that need to be corrected include: mean anomaly angle, longitude of ascending node, orbital inclination and satellite clock error at the reference time. Among them, orbital inclination and satellite clock error are both values representing the reference time in the ephemeris, and have their own rates of change. Although they are constants in the almanac, it will be more accurate to convert them to the reference time of the almanac. Although the longitude of ascending node represents its value at the start time of the week, which seems to be unrelated to the reference time, but according to the calculation formula of the longitude of ascending node at time t, these two values in almanac and ephemeris will be different when there is a rate of change of the longitude of ascending node.

If the longitude of ascending node needs to be the same at the same time t, then the following equation is used:

$$\Omega_0^a + (\dot{\Omega} - \dot{\Omega}_e)(t - t_{oa}) - \dot{\Omega}_e t_{oa} = \Omega_0^e + (\dot{\Omega} - \dot{\Omega}_e)(t - t_{oe}) - \dot{\Omega}_e t_{oe}$$
$$\Omega_0^a + (\dot{\Omega} - \dot{\Omega}_e)t - \dot{\Omega} t_{oa} + \dot{\Omega}_e t_{oa} - \dot{\Omega}_e t_{oa} = \Omega_0^e + (\dot{\Omega} - \dot{\Omega}_e)t - \dot{\Omega} t_{oe} + \dot{\Omega}_e t_{oe} - \dot{\Omega}_e t_{oe}$$
$$\Omega_0^a - \dot{\Omega} t_{oa} = \Omega_0^e - \dot{\Omega} t_{oe}$$
$$\Omega_0^a = \Omega_0^e + \dot{\Omega}(t_{oa} - t_{oe})$$

The equation shows how to correct the longitude of ascending node at week epoch.

The above calculations are universal for GPS, Galileo and BDS non-GEO satellites.

For GLONASS satellites, due to the different format of ephemeris and almanac parameters, different calculation methods are used from other satellite systems. Since the reference time of the GLONASS satellite almanac is the time of the satellite's first ascending node on the day, it is necessary to find an ephemeris that is as close to the ascending node as possible. This step is calculated in the following way: For the almanac of the target date, find the first ephemeris corresponding to that date. It should be noted that since the reference time of GLONASS differs from that of other satellite systems by about 3 hours, considering that the reference time of the ephemeris is usually at 15 minutes and 45 minutes of each hour, the reference time of the first ephemeris of GLONASS organized according to the UTC date is usually 11700 seconds. Because the orbit of the GLONASS satellite is an approximately inclined circular orbit, the position and velocity of the z coordinate are approximated as sine and cosine curves that vary with time, respectively. By calculating the four-quadrant inverse tangent, the approximate time of the next ascending node can be predicted, and the ephemeris closest to that time can be extracted for next step calculation.

After obtaining the ephemeris close to the ascending node time, in order to obtain the precise ascending node time, the position where the z coordinate is 0 can be calculated iteratively. At this time, the longitude of ascending node can be obtained through the x and y coordinates. Next, the orbital parameters can be calculated based on the two-body motion equation. Here, the speed calculated using the GLONASS ephemeris needs to be the speed in the inertial coordinate system. The steps to calculate the orbital elements using position and velocity are as follows:

Calculate the areal velocity vector: $\boldsymbol{h} = \boldsymbol{r} \times \dot{\boldsymbol{r}} = \begin{pmatrix} y\dot{z} - z\dot{y} \\ z\dot{x} - x\dot{z} \\ x\dot{y} - y\dot{x} \end{pmatrix}$

The inclination can be calculated: $i = \arctan(\frac{\sqrt{h_x^2 + h_y^2}}{h_z})$

The areal velocity can further be used to derive the semi-latus rectum $p = \frac{h^2}{GM}$ and then derive the semi-major axis $a = \left(\frac{2}{r} - \frac{v^2}{GM}\right)^{-1}$.

The orbital period can be calculated from the semi-major axis $T_{orbit} = \frac{2\pi}{n} = \frac{2\pi}{\sqrt{GM/a^3}}$.

Here, the period should be corrected according to GLONASS ICD 3.2 using the following equation:

$$T_{Alm} = T_{orbit} \cdot \left\{1 + \frac{3}{2} C_{20} \left(\frac{A_e}{p}\right)^2 \left[\left(2 - \frac{5}{2} \cdot sin^2 i\right) \cdot \frac{(1-e^2)^{3/2}}{(1+e \cdot cos\omega)^2} + \frac{(1+e \cdot cosv)^3}{1-e^2}\right]\right\}$$

Since the converted almanac itself does not need to be very accurate, in the above formula, approximate values $e \approx 0$ and $i \approx 65°$ can be used to have the part in the square brackets be treated as a constant.

Since the current time of ascending is predicted forward from the first ephemeris of current day, there might be possibility that this is the second ascending time of that day in GLONASS time. So it is necessary to compare current time of day and the orbit period to make corresponding judgments and adjust the ascending node time and longitude of ascending node accordingly.

The eccentricity can be calculated from the semi-latus rectum and semi-major axis: $e = \sqrt{1 - \frac{p}{a}}$.

Next to solve eccentric anomaly: $E = \arctan\left(\frac{\boldsymbol{r} \cdot \dot{\boldsymbol{r}}}{(1 - r/a) a^2 n}\right) = \arctan\left(\frac{\boldsymbol{r} \cdot \dot{\boldsymbol{r}}}{(1 - r/a)\sqrt{GM \cdot a}}\right)$.

Because at this time, the argument of latitude $u = 0$, so the argument of perigee ω equals the opposite value of true anomaly v, which results the equation $\omega = -\arctan(\frac{\sqrt{1-e^2} sinE}{cosE - e})$.

At this point, all orbital parameters of the GLONASS almanac have been calculated.

For BDS GEO satellites, since the custom reference system and the ECEF coordinate system not only have a 5-degree inclination difference on the x-axis, but are also inertial systems, the angle difference with ECEF on the z-axis is not fixed. Therefore, it is difficult to directly convert the orbital parameters in the custom reference system to the orbital parameters in the ECEF coordinate system. When converting the ephemeris of the GEO satellite to the almanac, first find a set of ephemeris parameters that are closest to the almanac reference time, and calculate the satellite position and velocity at the ephemeris reference time. The following calculation steps are similar to those of GLONASS. The orbital parameters at this moment are inferred from the position and velocity according to the equation of motion, and then the orbital

parameters are converted from the reference time of the ephemeris to the reference time of the almanac. When calculating orbital parameters, there are the following differences to the steps of GLONASS almanac conversion:

Firstly, since the reference time is not the ascending node time, the longitude of ascending node needs to be calculated through the areal velocity vector and convert to the epoch of week start: $\Omega_0 = atan2(h_x, -h_y) + \dot{\Omega}_e t_{oe}$

Secondly, mean anomaly at reference time need to be calculated using eccentric anomaly: $M_0 = E - e \cdot sinE$

Thirdly, because the reference time does not equal to the time of ascending, the argument of latitude should be calculated with $u = atan2(z \cdot h, y \cdot h_x - x \cdot h_y)$. In the above equation, h is the length of areal velocity vector $\boldsymbol{h}$.