Ginan

v1.0-alpha

Geoscience Australia and Frontier-SI

Published by Geoscience Australia and Frontier-SI

Contents

List of Figures

List of Tables

Welcome

Ginan is the fifth-brightest star in the Southern Cross (Epsilon Crucis) . It represents a red dilly-bag filled with special songs of knowledge. Indigenous Australians often used songs to convey and to pass on knowledge to others, song were also often used as a way to navigate the country.

We hope that you find this software tool kit will convey our understanding on how to process GNSS signals and will also help you to navigate the country!

the story Ginan was found by Mulugurnden (the crayfish), who brought the red flying foxes from the underworld to the sky. The bats flew up the track of the Milky Way and traded the spiritual song to Guyaru, the Night Owl (the star Sirius). The bats fly through the constellation Scorpius on their way to the Southern Cross, trading songs as they go.

The song informs the people about initiation, which is managed by the stars in Scorpius and related to Larawag (who ensures the appropriate personnel are present for the final stages of the ceremony).

The brownish-red colour of the dilly bag is represented by the colour of Epsilon Crucis, which is an orange giant that lies 228 light years away.

Part I

Overview

Introduction to Ginan GNSS Processing Toolkit

Ginan[1] is a collection of source code that is currently made up two distinct software repositories, the POD and the PEA. Using the POD and PEA together will allow you to estimate your own satellite orbits from a global tracking network. The POD (precise orbit determination) contains all of the source code needed to determine a GNSS satellite's orbit. You can establish the initial conditions of an orbit from a broadcast ephemeris file, or from an IGS SP3 file. It can then estimate it's own orbital trajectory based upon the models specified in configuration files, and output an SP3 file, or provide a partial files which can then be updated from a tracking network.

The PEA (parameter estimation algorithm) takes raw observations in RINEX format or in RTCM format, to estimates the parameters you are interested in. You can run it a single user mode, taking in orbit and clocks supplied by real-time streams to SP3 files obtained from the IGS to estimate your own position in static and kinematic mode. You can also run the PEA in a network mode, and take in a global network of observations to determine your own orbits and satellite clocks to support your application.

The software is aimed at supporting Australia's implementation of a national positioning infrastructure that supports the objective of 'instantaneous GNSS positoning anywhere, anytime, with the highest possible accuracy and the highest possible integrity.

Carrier phase ambiguity. The underlying signals transmitted by the Global Navigation Satellite Systems (GNSS) can be considered as waves, just like repeating sine waves from high school mathematics. Measurements of these waves are referred to as carrier phase observations, and they are used to provide the precise distance, with mm precision and accuracy, between the orbiting satellites and user's receiver that are subsequently used to compute position. However, a complicating factor is that carrier phase observations have an ambiguous component where the total whole number of waves, or integer cycles, between the satellite and the user's receiver cannot be measured, only the fractional part. The unknown number of integer cycles is called the carrier phase ambiguity. Fortunately, the ambiguities can be estimated, and the mathematical and statistical solution to this problem is known as integer ambiguity estimation. While there is a long history of research in this area, which has largely focused on GPS applications, the most optimal solution to this problem when simultaneously combining data from all the GNSS remains unresolved.

Atmosphere delay of GNSS signals. The Earth is surrounded by layers of gases held by Earth's gravity. Signals, such as those transmitted by GNSS, propagated from space are delayed as they pass through the atmosphere. In the troposphere, the region from the Earth's surface to approximately 20 km altitude, the delay is proportional to temperature, pressure and humidity. The ionosphere, the region from 50-1000 km altitude, causes delay as a function of the frequency of the signal. The composition of both the troposphere and ionosphere vary both in space and time, and this variability currently limits the accuracy, speed and reliability of positioning. But it's not all bad news, and like a CAT scan in medical science, the new GNSS signals and satellites can potentially be combined to provide a more complete three dimensional picture of the atmospheric delay as a function of time. Models that more completely remove the nuisance atmospheric signals will lead to improved accuracy, speed and reliability of positioning.

Precise Point Positioning (PPP) and Real Time Kinematic (RTK). Conventional positioning technologies almost exclusively use a technique called Real Time Kinematic (RTK). The RTK technique takes information from nearby Continuously Operating Reference Stations (CORS) to generate corrections for measurements made by users. One of the most important of these is the carrier phase ambiguity correction. The ability to correctly determine carrier phase ambiguities with RTK is determined by many factors, such as the distance between the reference stations and the user and also atmospheric

[1] https://github.com/GeoscienceAustralia/ginan

effects. Consequently, RTK relies on relatively dense CORS networks with a typical spacing of 30 to 70 km. An alternative to RTK is Precise Point Positioning (PPP). The PPP technique, rather than directly using measurements from nearby reference stations, uses global satellite orbit and clock information such as that provided by the International GNSS Service. The major advantage of PPP is that it doesn't require a dense CORS network nearby the user's location just access to global products. Unfortunately, the PPP technique can have difficulties in resolving carrier phase ambiguities in real time, but additional research focused on greater exploitation of multi-GNSS data and more regional approaches may in the future overcome this limitation.

Installation

3.1 To Install

In this section we will describe how to install the PEA and POD from source. An alternative option to installing all of the dependencies and the source code would be to use one of our docker images available from Docker Hub. Instructions on how to do this are in (see Docker).

3.1.1 PEA

Dependencies the following packages need to be installed with the minimum versions as shown below. This guide will outline the preferred method of installation.

CMAKE ¿ 3.0 requires openssl-devel to be installed (requires openssl-devel) YAML ¿ 0.6 Boost ¿ 1.70 gcc ¿ 4.1 Eigen3 Build To build the PEA Precise Estimation Algorithm...

We suggest using the following directory structure when installing the Ginan toolkit. It will be created by following this guide.

The following is an example procedure to install the dependencies necessary to run the pea on a base ubuntu linux distribution

Update the base operating system:

```
$ sudo apt update
$ sudo apt upgrade
```

Install base utilities gcc, gfortran, git, openssl, blas, lapack, etc

```
$ sudo apt install -y git gobjc gobjc++ gfortran libopenblas-dev openssl curl net-tools
    openssh-server cmake make \
liblapack-dev gzip vim libssl1.0-dev python3-cartopy python3-scipy python3-matplotlib
    python3-mpltoolkits.basemap
Create a temporary directory structure to make the dependencies in:
$ sudo mkdir -p /data/tmp
$ cd /data/tmp
```

YAML We are using the YAML library to parse the configuration files used to run many of the programs found in this library (https://github.com/jbeder/yaml-cpp). Here is an example of how we have installed the yaml library from source:

```
$ cd /data/tmp
$ sudo git clone https://github.com/jbeder/yaml-cpp.git
$ cd yaml-cpp
$ sudo mkdir cmake-build
$ cd cmake-build
$ sudo cmake .. -DCMAKE\_INSTALL\_PREFIX=/usr/local/ -DYAML\_CPP\_BUILD\_TESTS=OFF
$ sudo make install yaml-cpp
$ cd ../..
$ sudo rm -fr yaml-cpp
```

Boost We rely on a number of the utilities provided by boost (https://www.boost.org/), such as their time and logging libraries.

```
$ cd /data/tmp/
$ sudo wget -c https://dl.bintray.com/boostorg/release/1.73.0/source/boost_1_73_0.tar.gz
$ sudo gunzip boost_1_73_0.tar.gz
$ sudo tar xvf boost_1_73_0.tar
$ cd boost_1_73_0/
$ sudo ./bootstrap.sh
$ sudo ./b2 install
$ cd ..
$ sudo rm -fr boost_1_73_0/ boost_1_73_0.tar
```

Eigen3 is used for performing matrix calculations, and has a very nice API.

```
$ cd /data/tmp/
$ sudo git clone https://gitlab.com/libeigen/eigen.git
$ cd eigen
$ sudo mkdir cmake-build
$ cd cmake-build
$ sudo cmake ..
$ sudo make install
$ cd ../..
$ sudo rm -rf eigen
Installing PEA
PEA Executable
$ cd /data/acs/
```

Clone the repository via https:

```
$ git clone https://bitbucket.org/geoscienceaustralia/pea.git
```

You should now have...

Prepare a directory to build in, its better practise to keep this separated from the source code.

```
$ cd pea/cpp
$ mkdir -p build
$ cd build
```

Run cmake to find the build dependencies and create the makefile. You have the choice of adding in a couple of compile options. Using the flag -DENABLE_MONGODB=TRUE will set up the mongodb utilities, adding the flag -DENABLE_OPTIMISATION=TRUE will set up the compiler to run optimisation O3. Enabling the optimisation flag will speed up the processing by a factor of 3, however this can lead to compile errors depending on the system you are compiling on, if this happens remove this option.

```
$ cmake ..
or to enable MONGODB utilities
$ cmake -DENABLE_MONGODB=TRUE ..
and to enable Optimisation
$ cmake -DENABLE_MONGODB=TRUE -DENABLE_OPTIMISATION=TRUE ..
```

Now build the pea

```
$ cmake --build $PWD --target pea
```

To change to build location substitute your preferred destination for $PWD , e.g /usr/local/bin
Alternatively to the command above you can make the code in parallel using:

```
$ make -j 5 all
```

where the -j flag controls how many jobs can be run at the same time.
Check to see if you can execute the pea:

```
$ ./pea
```

and you should see something similar to:

```
PEA starting...
Options:
  --help                      Help
  --verbose                   More output
  --quiet                     Less output
  --config arg                Configuration file
  --trace_level arg           Trace level
  --antenna arg               ANTEX file
  --navigation arg            Navigation file
  --sinex arg                 SINEX file
  --sp3file arg               Orbit (SP3) file
  --clkfile arg               Clock (CLK) file
  --dcbfile arg               Code Bias (DCB) file
  --ionfile arg               Ionosphere (IONEX) file
  --podfile arg               Orbits (POD) file
  --blqfile arg               BLQ (Ocean loading) file
  --erpfile arg               ERP file
  --elevation_mask arg        Elevation Mask
  --max_epochs arg            Maximum Epochs
  --epoch_interval arg        Epoch Interval
  --rnx arg                   RINEX station file
  --root_input_dir arg        Directory containg the input data
```

```
  --root_output_directory arg Output directory
  --start_epoch arg           Start date/time
  --end_epoch arg             Stop date/time
  --dump-config-only          Dump the configuration and exit
PEA finished
```

The documentation for the pea can be generated similarly using doxygen if it is installed.

```
$ sudo apt-get install doxygen
$ cd pea/cpp/build
$ make doc_doxygen
```

The docs can then be found at doc_doxygen/html/index.html

3.1.2 POD from source

Dependencies

The open basic linear algebra library (Openblas.x86_64,liblas-libs.x86_64) (You may need to run the command ln -s /usr/lib64/libopenblas.so.3 /usr/lib64/libopenblas.so) A working C compiler (gcc will do), a working C++ compiler (gcc-g++ will do) and a fortran compiler (we have used gfortran) Cmake (from cmake.org) at least version 2.8 If the flags set in CMakeLists.txt do not work with your compiler please remove/replace the ones that don't

Build To build the POD ...

```
$ cd pod
$ mkdir build
$ cd build
$ cmake3 ..
$ make >make.out 2>make.err
$ less make.err (to verify everything was built correctly)
```

You should now have the executables in the bin directory: pod crs2trs brdc2ecef

Test To test your build of the POD ... - You may not need the ulimit command but we found it necessary

```
$ cd ../pod/test
$ ulimit -s unlimited
$ ./sh_test_pod
```

At the completion of the test run, the sh_test_pod script will return any differences to the standard test resuts

Configuration File The POD Precise Orbit Determination (./bin/pod) uses the configuration file:

PEA examples

In this section we go through a number of different ways that the pea can be used to process GNSS data.

1. Precise Point Positioning (PPP) processing - In this section we will demonstrate how to processing in PPP mode using the Ionosphere free combination, we will provide an example on how to use IGS products to obtian a float solution, and then an example on how to obtain an ambguity fixed solution. We will also cover how to process gnss streams in realtime.
2. Obtain an orbit solution from a global tracking network
3. Obtain an orbit and clock solution from a global tracking network
4. How to process a Global solution in real-time
5. How to obtain an ionosphere model

POD Examples

5.1 Processing Example 1

In this example the pod will perform a dynamic orbit determination for PRN04 over a 6 hour arc. The full gravitational force models are applied, with a cannonball model SRP model.

To run the POD ...

```
$ bin/pod
```

This should output the following to stdout...

```
Orbit Determination
Orbit residuals in ICRF : RMS(XYZ)   1.6754034501980351E-002   5.2908718335411935E-002
     1.5676115599034774E-002
Orbit Determination: Completed
CPU Time (sec)   298.48134399999998
External Orbit comparison
Orbit comparison: ICRF
RMS RTN   2.8094479714173427E-002   2.4358145601708528E-002   4.4097979280889953E-002
RMS XYZ   1.6754034501980351E-002   5.2908718335411935E-002   1.5676115599034774E-002
Orbit comparison: ITRF
RMS XYZ   3.9069978513805753E-002   3.9343671258381237E-002   1.5660654272651970E-002
Write orbit matrices to output files
CPU Time (sec)   349.19307899999995
The results above show that our orbits arcs, over 6 hours, are currently within 2-5 cm
    of the final combined IGS orbit.
```

The processing also produces the following output files... /beginverbatim /endverbatim

5.2 Processing Example 2 - ECOM2 SRP

In this example we will change the SRP model to use the ECOM2 model.

Edit the EQM.in file so that the Solar Radiation Pressure configuration section now looks:

! Solar Radiation Pressure model: ! 1. Cannonball model ! 2. Box-wing model ! 3. ECOM (D2B1) model SRP_model 3

Then edit VEQ.in, so that the Non-gravitational forces now looks like:

! Solar Radiation Pressure model: ! 1. Cannonball model ! 2. Box-wing model ! 3. ECOM (D2B1) model SRP_model 3

run the POD ...

```
$ bin/pod
```

This should output the following to stdout...

```
Orbit Determination
Orbit residuals in ICRF : RMS(XYZ)   2.0336204859568077E-002   8.4715644601919167E-003
     3.9687932322714677E-002
Orbit Determination: Completed
CPU Time (sec)   299.68054799999999
External Orbit comparison
Orbit comparison: ICRF
RMS RTN   2.8182836396022540E-002   2.4598832384842121E-002   2.5879201921952168E-002
RMS XYZ   2.0336204859568077E-002   8.4715644601919167E-003   3.9687932322714677E-002
Orbit comparison: ITRF
RMS XYZ   1.8757217704973204E-002   1.1635302426688266E-002   3.9702619816620370E-002
Write orbit matrices to output files
CPU Time (sec)   350.88653299999999\
```

5.3 Example 3 - (pod/examples/ex3)

GPS IGS SP3 file orbit fitting, orbit prediction and comparison to next IGS SP3 file

5.4 Example 4 - (pod/examples/ex4):

Integration of POD initial conditions file generated by the PEA

5.5 Example 5 - (pod/examples/ex5):

ECOM1+ECOM2 hybrid SRP model In each example directory (ex1/ex2/ex3/ex4) there is a sh_ex? script that when executed will run the example and compare the output with the expected solution.

The POD is designed to do some stuff.

Part II

Background Theory

GNSS Overview

6.1 GPS

6.2 Glonass

6.3 Galileo

6.4 Beidou

6.5 QZSS

The and POD is designed to do some stuff.

Observation Modelling

7.1 Ionosphere-Free Observations

7.2 Undifferenced - Uncombined

To do .. *The PEA and POD* is designed to do some stuff.

7.3 GPS Quarter Cycle

The L_2 Civil code (L2C-code) is shifted by a quarter cycle with respect to the P-code. If a receiver is using either the one or the other code to reconstruct the carrier phase measurements, then they will also be shifted by a quarter of a wavelength relative to each other.
The noise of phase measurement based on L2C-code is usually lower than the noise of phase observations reconstructed based on p-code. However this is only possible to obtain from satellites that have launched since GPS Block IIR-M was deployed. Older GPS satellites do not provide the L2C-code signal.
In RINEX version 2, in order to prevent potential problems in ambiguity resolution some receiver manufactures do correct the phase measurements by 0.25 cycles to keep the phase observations consistent, while others provide the uncorrected phase measurements, an din other cases the user can decide if the correction is applied. Since RINEX version 3.02 the definition has been defined that the phase measurements need to be corrected to have a consistent set of observable that can be introduce for ambiguity resolution.
One option to prevent having the quarter cycle issue is to ensure that ambiguities are not resolved between a Block IIR-M or later with an older satellite.

7.4 Single-satellite and Single-Receiver Observation Combinations

Linear combinations of the original observations are often applied to eliminate model parameters (eg ionosphere slant delays) or to transform ambiguity parameters (eg widelane transformation). Some of these transformations maintain the information content of the system as they are invertible, but other transformation that are used are do not, and these should be used with caution as the information that contributes to the parameters of interest are lost.
A large number of different permutations of linear combinations can be formed, with just the dual-frequency legacy signals, this can increases even more as more additional frequencies are available with modernized GNSS signals are deployed. We will only cover the ones used by the PEA in this manual.
Combining two or more carrier-phase observations into a new signal leads to a different frequency/wavelength. To generalise in the case of a combination for which $\sigma\alpha = 1$, the combined frequency f_c is:

$$f_c = \sigma_{j=1}^{n} i_j f_j \tag{7.1}$$

Remembering that all frequencies for GNSS are derived from a single frequency f_0 by multiplication with an integer k_j, the individual frequency is obtained from $f_j = j_j f_0$. substituting this into the above equation

$f_c = (\sigma_{j=1}^{n} i_j k_j) = k f_0$

where k is the *lane number*. The corresponding wavelength is:

$\lambda_c = c/k f_0 = \lambda_0/k$

where λ_0 is the wavelength of the base frequency f_0. Since all i_j and k_j are integers, k is also an integer. This parameter uniquely defines the frequency and wavelength of the new signal combination.

With the help of the lane number k, the combinations can be categorized into three groups:

1. *wide-lane* combination,where the combined wavelength is larger than the largest individual wavelength in the combination.

2. *intermediate* combinations, for which λ_0 leas between the largest and shortest individual wavelength.

3. *narrow-lane* combinations which have a shorter wavelength than the individual signal with the shortest wavelength in the combination.

The zero-differenced, ionosphere free mathematical model for code and phase measurements using dual-frequency can be described by:

$$E(P^S_{r,IF}) = \rho^s_r + c(dt_r - dt^s) + \tau^s_r + c(d_{r,IF} + d^s_{r,IF})$$

$$E(L^S_{r,IF}) = \rho^s_r + c(dt_r - dt^s) + \tau^s_r + \lambda_{IF}(z^s_{r,IF} + \delta_{r,IF} - \delta^S_{IF}$$

with:
$E()$ the expectation notation $P^S_{r,IF}$ code measurements (m) between satellite s and receiver r for the ionosphere-free measurements;
$L^S_{r,IF}$ phase measurements (m)
ρ the geometric distance (m)
c speed of light (m/s)
dt_r receiver clock error (s)
dt^S satellite clock error (s)
τ^s_r slant troposphere delay between satellite s and receiver r (m)
$dt_{r,IF}$ receiver code bias (s)
dt^S_{IF} satellite code bias (s)
λ_{IF} ionosphere-free wavelength (m)
$z^S_{r,IF}$ ionosphere-free ambiguity vector (cycle)
$\delta^S_{r,IF}$ receiver ionosphere-free phase bias (cycle)
δ^S_{IF} satellite ionosphere-free phase bias (cycle)
The ionosphere-free combination will remove the first-order ionosphere delay, both equations are rand deficient and the ambiguity term in eq (2) is not an integer. In order to resolve the ambiguities the linear dependency among the unknown parameters needs to be resolved and the integer property of the ambiguities needs to be recovered.

7.5 Widelane combinations

The group of wide-lane combinations is especially useful to help with integer ambiguity resolution due to their longer wavelength.

7.5.1 Common widelane

To improve ambiguity resolution, often the wide lane linear combination (also called L5 or LW linear combination) is formed from the L1 and L2 carrier phase observables. It is designed to be a geometry-preserving combination using only carrier-phase measurements in the frequencies f_a and f_b. By selecting the integer coefficients as $i_A = +1$ and $i_B = -1$, one obtains the WL carrier-phase combination $\phi^s_{r,WL}$ in units of cycles:

$$\phi^s_{r,WL} = \phi^A_{r,A} - \phi^S_{r,B} = \frac{\psi^s_{r,A}}{\lambda_A} - \frac{\psi r, B^s}{\lambda_B} \tag{7.2}$$

The corresponding wavelength λ_{WL} is:

$$\lambda_{WL} = \frac{c}{f_A - f_B} \tag{7.3}$$

Multiplication of the two above equations leads to the wide-lane combination $\psi^s_{r,WL}$ in units of meters:

$$\psi^S_{r,WL} = \frac{f_A}{f_A - f_B}\psi^s_{r,A} - \frac{f_B}{f_A - f_B}\psi^s_{r,B} \tag{7.4}$$

7.5.2 Melbourne-Wubbena linear combination

The Melbourne-Wubbena linear combination (Melbourne 1985),(Wubbena 1985) is a linear combination of the L1 and L2 carrier phase plus the P1 and P2 pseudorange. The geometry, troposphere and ionosphere are eliminated by it. The Melbourne-Wubbena linear combination can be represented as:

$E(L^S_{r,IF}) - \frac{cf_2 z^s_{r,w}}{f_1^2 - f_2^2} = \rho^s_r + c(dt_{r,IF} - dt^s_{IF}) + \tau^s_r + \lambda_n z^s_{r,1} + (\lambda_{IF}\delta_{r,IF}$

Since „elevation measurements to avoid the multipath impacts from the code observation. Normally, with 30 degree elevation cut-off, an averaging of 5 minutes of (4) is good enough to fixing the wide-lane ambiguities [RD 04]. The rests are the wide-lane phase bias, which can be broadcasted to the user for user side wide-lane ambiguity resolution. Either choosing a pivot receiver bias or a single-differencing between two satellites can avoid the linear dependency.

-doesn't need lambda not as correlated

Kalman Filtering

The Kalman filter is over 50 years old but is still one of the most important and common data fusion algorithms in use today. Named after Rudolf E.Kálmán, the great success of the Kalman filter is due to its small computational requirement, elegant recursive properties, and its status as the optimal estimator for one-dimensional linear systems with Gaussian error statistics. Typical uses of the Kalman filter include smoothing noisy data and providing estimates of parameters of interest. Kalman filtering is used in a wide range of applications include global positioning system receivers, in control systems, through to the smoothing the output from laptop trackpads, and many more.

8.1 Overview of Kalman Filtering

Kalman filter are typically used to estimate parameters which change with time. Parameters with no process noise are called deterministic. A Kalman filter has measurements $y(t)$, with noise $v(t)$, and a state vector (or a parameter list) which have specified statistical properties.

The observation equation at time t:

$$y_t = A_t x_t + v_t \tag{8.1}$$

The state transition equation:

$$x_{t+1} = S_t x_t + w_t \tag{8.2}$$

Covariance matrices

$$< v_t v_t^T >= V_t, < w_t w_t^T >= W_t \tag{8.3}$$

The kalman filter processing is broken up into three main steps.

Prediction uses a process noise model to 'predict' the parameters at the next data epoch, subscript is time quantity refrs to, where as the superscript refers to the time of the data:

$$\hat{x}_{t+1}^t = S_t \hat{x}_t^t \tag{8.4}$$

where, S_t is the state transition matrix

$$C_{t+1}^t = S_t C_t^t S_t^t + W_t \tag{8.5}$$

where, w_t is the process noise covariance matrix. The state transition matrix S projects the state vector (parameters) forward to the next epoch.

- For random walk $S = 1$
- For rate terms: S is matrix $[1 \delta t][01]$
- for FOGM: $S = e^{-\delta t \beta}$
- For white noise $S = 0$

The second equation projects the covariance matrix of the state vector, C, forward in time. Contributions from the state transition and process noise (W matrix). W elements are 0 for deterministic parameters. *The Kalman gain* is the matrix that allocates the differences between the observation at time t+1 and their predicted value at this time based on the current values of the state vector according to the noise in the measurements and the state vector noise.

8.2 Comparison between Weighted Least Squares and Kalman Filtering

- In kalman filtering apriori constraints must be give for all parameters. This is not needed in weighted least squares, but can also be done.
- Kalman filters can allow for 0 variance parameters, this cannot be done in WLS, as this requires the inversion of the constraint matrix.
- Kalman filter can allow for a method of applying absolute constraints, WLS can only tightly constrain parameters.
- Kalman filters are more prone to numerical stability problems, and take longer to run (they have more parameters).
- Process noise models can be implemented in WLS, but they are computationally slow.

8.3 Implementation in the PEA

8.3.1 Observation Weighting

The GNSS observations can be weighted in three different ways in the PEA:

- Unity - all observations are assigned the same variance
- Elevation Dependent - an elevation dependent function is used to scale the observations, those at higher elevation are given more weight (a smaller standard deviation) than those observed at lower elevation
- SNR observations - the Carrier to Noise observations supplied by the receiver are used to determine the observation weight. Generally speaking this is very similar to elevation weighting, but is useful when use observations obtained from a non-geodetic grade receiver/antenna.

8.3.2 Unity

8.3.3 Elevation Dependent Weighting

8.3.4 SNR weighting

8.3.5 Process Noise

Currently in the PEA we have Gaussian, and random walk process noise models implemented.

The units are in meters, and they are given as σ sqrt(variance)

For a random walk process noise, the process noise is incremented at each epoch as $\sqrt{process\_noise} * dt$ where dt is the time step between filter updates.

If you want to allow kinematic processing, then you can increase the process noise e.g. proc_noise [0.003] proc_noise_dt: second

equates to $\frac{0.003}{/}\sqrt{s}$

Or if you wanted highway sppeds 100km/hr = 28 m/s proc_noise [28] proc_noise_dt: second proc_noise_model: Gaussian or proc_noise_model: RandomWalk or proc_noise_model: FOGM 1.0 Where the number after FOGM gives the correlation time B That way we could keep the same: proc_noise: [0.01] proc_noise_dt: hour

A nice value for using VMF as an apriori value is 0.1mm /sqrt(s)

```
trop:
    estimated:          true
    sigma:              [0.1]
    proc_noise:         [0.01]
    proc_noise_dt:      hour
```

8.4 Recommended Reading

1. https://ocw.mit.edu/courses/earth-atmospheric-and-planetary-sciences/12-540-principles-of-the-global-positioning-system-spring-2012/lecture-notes/MIT12_540S12_lec13.pdf

RTS Smoothing

While the Kalman filter is an optimal solution to computing state estimates from all previous data, better estimates could be obtained if all future data were also incorporated.

The RTS Smoothing algorithm is an approach to determine estimates of states and uncertainties by considering the state transistion between two kalman filtered estimates, smoothing the transition between prior and 'future' data.

9.1 Example

Consider the system of a single particle moving in one dimension. At time t=0, the particle's position is measured to be x=0. The system then evolves with a random walk, with no more measurements until the time t=100.

At t=99, the particles position has a large uncertainty - it has likely moved from its original position, however, with no more data available, its mean expected value remains as x=0.

At t=100, the particles position is again measured, this time as x=10. If this system were monitored using a Kalman filter, the expected position of the particle would be a constant x=0 for the first 99 seconds, before an abrupt change in location at t=100. During the 100 seconds, the variance would increase steadily, before abruptly returning to a low value when the second measurement is taken.

If the kalman filter measurements were taken in reverse order, with the first measurement at t=100, the variance of the particle would steadily increase going backward in time until t=0, before the second measurement again reduced the variance, this time at t=0.

Using an RTS Smoother effectively combines both forward and backward filtering. At t=1, the variance is low due to the measurement at t=0. Likewise, at t=99, the variance is low due to the measurement at t=100. In this example, in addition to the measurements at t=0 and t=100, the expected mean value at t=50 can be expected to be the midpoint between the two measurements - a result that can not be obtained using Kalman filtering alone.

In order to accurately calculate the expected position and variance at t=50 however, knowledge of the measurement at t=100 was required (50 seconds later). RTS smoothed estimates necessarily lag behind the primary Kalman filter to allow some time for future data to be obtained. The length of this lag determines the effectiveness of the smoothing.

9.2 Usage

All kalman filters in the toolkit are capable of having RTS Smoothing applied. If configured appropriately with an RTS_lag and output files, intermediate filter results will be stored to file for reverse smoothing.

For real-time processing, a small lag may be applied to improve short-term accuracy. After each filtering stage, the new result is propagated backward through time to correct the previous N epochs. Each epoch worth of lag however requires a comparable processing time to an Kalman filter processing stage - a lag of N epochs may slow processing by up to a multiple of N.

For post-processing, the optimal lag is to use all future and past data. This is achieved by first computing the forward solution, before propagating the final results backward through to the first epoch. The processing time required for a complete backward smoothed filter may be less than 2x a non-smoothed filte - considerably faster than a finite lag in real-time.

Orbit Modelling

POD is designed to do some stuff.

10.1 Gravitational Force Models

10.2 Non-Gravitational Force Models

10.2.1 Solar Radiation Force Models

The magnitude of the SRP acting on the satellite depends on a wide range of parameters. The distance to the Sun and the position of the satellite with respect to Earth and Sun (regarding possible eclipses) define the intensity of the incoming radiation. The geometry of the satellite, the optical properties of the external surfaces, and the actual orientation with respect to the Sun largely influence the orientation and magnitude of the evolving SRP.

10.2.2 Cannonball

The most basic approach with regard to its analytic development is referred to as the cannonball model.The cannonball model provides a useful, first-order approximation, however, due to its homogeneous material properties and symmetrical shape approximation. We recommend its use as an apriori model, before estimation.

10.2.3 ECOM I

10.2.4 ECOM II

10.2.5 ECOM C

10.2.6 Box Wing

10.2.7 Antenna Thrust

10.2.8 Albedo

10.3 Transformtation between Celestial and Terrestrial Reference Systems

The variational equations obtained from the POD need to be transformed into the terrestrial reference frame so that the adjustments can be made in the ECEF frame that the PEA operates in.

$$[CRS] = Q(t)R(t)W(t)[TRS] \tag{10.1}$$

Where, CRS is the Celestial Reference System TRS is the Terrestrial Reference Systems $Q(t)$ is the Celestial Pole motion (Precession-Nutation) matrix $R(t)$ is the Earth Rotation matrix $W(t)$ is the Polar motion matrix

$$Q(t) = \begin{bmatrix} 1 - aX^2 & -aXY & X \\ -aXY & 1 - aY^2 & Y \\ -X & -Y & 1 - a(X^2 + Y^2) \end{bmatrix} \tag{10.2}$$

$$R(t) = R_2(-\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\Theta & cos\theta & 0 \\ 0 & 01 & \end{bmatrix} \tag{10.3}$$

Ionosphere Modelling

Ionospheric delay is the most important nuisance parameter on GNSS processing. GNSS processing algorithm needs to account for it by estimating, correcting or cancelling its effects. The objective of the project is to develop Ionospheric modelling modules to aid in GNSS data processing. he modules will allow end user position algorithms correct the effect of ionosphere delay. The modules also allow the network processing more effectively estimate the ionospheric delays.
Two potential ionosphere models have been implemented to date. Both are dual-layer models, each layer containing an ionosphere density map. The maps are represented by spherical cap harmonics in one case and 3rd order b-splines in another. The effectiveness of the maps has been tested using measurements from 60 Australian stations distributed. The geometry free combination of carrier smoothed pseudorange have been used as ionosphere delay measurements. The spherical cap harmonics tested so far can represent the ionospheric delays with an accuracy of about 2-3 TECu, slightly better than the model used for IONEX (single layer, grid based map), but not enough for the 0.2 TECu accuracy needed for PPP. The specific parameters (number of layers, map resolution, layer heights and order of the base function) of each model will be refined to improve model accuracy.

Ambiguity Resolution

An advantage of the ambiguity fixed solutions is the significantly reduced number of parameters which have to be solved for. A reduction of the normal equation to be inverted is important because usually a duplication of its size leads to over four times longer computing time for the inversion. If many parameters are estimated (orbits, Earth orientation parameters etc.) ambiguity resolution improves also the results of much longer sessions than the traditional daily solution. Due to the FDMA technology the ambiguity resolution for GLONASS is not a straight forward task, and we have not attempted to implement this into the pea yet. There are many methods that can be used to resolve ambiguities, but they mainly consist of two steps:

1. The ambiguities are estimates as real numbers (with the other parameters).
2. Integer values of the ambiguities are resolved using the results of step 1 (the real-value ambiguities and the VCV matrix) employing a number of statistical tests to ensure a reliable estimate.

Determining a reference or a pivot is normally done by selecting a station with the most number of observations, however this is not possible to no apriori for a systems that is designed to run in real-time. In the pea we have implemented a number of different ambiguity resolution strategies:

- rounding and weighted rounding
- bootstrapping
- lambda decorelation
- BIE

12.1 rounding algorithm

The simplest strategy to apply is to round the real-values estimates to the nearest integers, without using any variance co-variance information.

12.2 bootstrapping

The bootstrapping algorithm takes the first ambiguity and rounds its value to the nearest integer. Having obtained the integer value of this first ambiguity, the real-valued estimates of all remaining ambiguities are then corrected by virtue of their correlation with the first ambiguity. Then the second, but now corrected, real-valued ambiguity estimate is rounded to its nearest integer, and the process is then repeated again with both ambiguities held fixed, and the process is continued until all ambiguities are accommodated. Thus the bootstrapped estimator reduces to 'integer rounding' in case correlations are absent.

12.3 lambda

12.4 BIE

12.4.1 Melbourne-Wubbena linear combination

The Melbourne-Wubbena linear combination (Melbourne 1985),(Wubbena 1985) is a linear combination of the L1 and L2 carrier phase plus the P1 and P2 pseudorange. The geometry, troposphere and

ionosphere are eliminated by it. The Melbourne-Wubbena linear combination can be represented as:

$$E(L^S_{r,IF}) - \frac{cf_2 z^s_{r,w}}{f_1^2 - f_2^2} = \rho^s_r + c(dt_{r,IF} - dt^s_{IF}) + \tau^s_r + \lambda_n z^s_{r,1} + (\lambda_{IF}\delta_{r,IF} \tag{12.1}$$

Since „% comprises of both code and phase measurements, it is reasonable to exclude the lower elevation measurements to avoid the multipath impacts from the code observation. Normally, with 30 degree elevation cut-off, an averaging of 5 minutes of (4) is good enough to fixing the wide-lane ambiguities [RD 04]. The rests are the wide-lane phase bias, which can be broadcasted to the user for user side wide-lane ambiguity resolution. Either choosing a pivot receiver bias or a single-differencing between two satellites can avoid the linear dependency. doesn't need lambda not as correlated

12.5 Narrowlane and phase clock estimation

highly correlated need lambda With the fixing of the wide-lane ambiguity, equation (1) and (2) can be further deducted as: The code bias and phase bias in equations (8.1) and (6) can be lumped into the corresponding receiver and satellite clock errors. Then equations (5) and (6) become: with: By such an reformulation, there are two types of satellite clock: 1) IGS type clock, but estimated only from code measurements; 2) phase clock, estimated using phase measurements and can be used to support PPP ambiguity resolution on the user side. The drawback of this approach is that there is no precise IGS compatible clock after the processing and it has to be derived from the existing PEA processing

12.6 Narrowlane and phase bias estimation

In this approach, equation (7) holds the same for the code measurements. However, in equation (8), the phase biases are not lumped into the clocks, but the ambiguities. More specifically, equation (8) can be written as: In equation (9), the clocks are the same as the clocks provided in equation (7), which means the precise IGS satellite clock can be estimated. The narrow-lane integer ambiguities, as shown in equation (9), are linearly dependent on the phase biases, which means they cannot be estimated simultaneously. However, similar as the wide-lane ambiguity resolution, the narrow-lane integer ambiguities can be fixed by rounding the float solution to the nearest integer, and the remaining will be the phase bias, which can be broadcasted to the user for user side PPP ambiguity resolution. *Bootstrapping* is designed to do some stuff. *Rounding Lambda* Decorrelation algorithm

12.7 References on Ambiguity Resolution

- A modified phase clock/bias model to improve PPP ambiguity resolution at Wuhan University Journal of Geodesy - Geng et al. (2019)
- On the interoperability of IGS products for precise point positioning with ambiguity resolution Journal of Geodesy - Simon et al. (2020)
- Resolution of GPS carrier-phase ambiguities in precise point positioning (PPP) with daily observations Journal of Geodesy - Ge et al. (2008)
- Real time zero-difference ambiguities fixing and absolute RTK ION NTM - Laurichesse et al. (2008)
- Undifferenced GPS ambiguity resolution using the decoupled clock model and ambiguity datum fixing Navigation – Collins et al. (2010)
- Improving the estimation of fractional-cycle biases for ambiguity resolution in precise point positioning Journal of Geodesy – Geng (2012).
- Modeling and quality control for reliable precise point positioning integer ambiguity resolution with GNSS modernization GPS Solutions – Li et al. (2014).

Minimum Constraints

When running the toolkit in network mode, the positions of receivers and satellites may both be estimated simultaneously. This presents a complication in that the absolute positions of all elements may be unconstrained - the model of the system would be completely consistent if every element of the network (receivers and satellites) were on the opposite side of the planet! For the results of such processing to be of value, the system needs to be referenced to a standard reference frame.

One method of ensuring the system is referenced to a suitable reference frame is to constrain receiver positions to their nominal position in the reference frame. Constraining 3 receivers is sufficient to ensure the system is well defined, but this gives select receivers precedence over all others - a movement of one of these receivers will instead show up as a movement of every other reciever on the planet.

An alternative method may be to weakly constrain all receivers to their nominal positions. This removes the priority effect of choosing 3 receivers, but will also apply a small restoring bias to the estimated position of the receivers.

Minimum constrains is a method of referring a system of estimates to a standard reference frame without unduly prioritising any receiver, or biasing the measurements.

13.1 Computation

To compute a minimally constrained system, the deviations of all receivers from their nominal positions are used as pseudo-observations in a filter to estimate a transformation between reference frames.

The filter estimates a rigid transformation comprising of 3d rotation and translation components, which when applied produces a least-squares solution of the errors of all desired receivers.

The estimated transformation is then applied to the network solution, transforming both the estimated position states, as well as the covariances associated with them.

Equation Conventions

In this manual we will be adhering to the following conventions

14.1 List of Symbols

- $\boldsymbol{i}$ Receiver identification or $\boldsymbol{r}$
- $\boldsymbol{j}$ Satellite identification or $\boldsymbol{s}$
- $\boldsymbol{k}$ Epoch number or $\boldsymbol{t}$
- $\boldsymbol{q}$ GNSS type (GPS,GALILEO,GLONASS,QZSS)
- $\boldsymbol{c}$ Speed of light [m/s]
- $\boldsymbol{x}$ Vector of parameters to be estimated [m]
- $\boldsymbol{y}$ Vector of observations [m]
- $\boldsymbol{A}$ Design matrix
- $\boldsymbol{\sigma}$ Standard deviation of observable
- $\boldsymbol{\Delta}$ Increment to a priori values [m]
- $\boldsymbol{\omega}$avelength or $\boldsymbol{\lambda_1, \lambda_2, \lambda_5}$
- $\boldsymbol{f_1, f_2, f_5}$ frequency
- $\boldsymbol{\alpha}$mbguity or $\boldsymbol{N}$ Real valued ambguity and $\bar{\boldsymbol{N}}$ Integer part of real valued ambguity
- $\boldsymbol{\alpha}$ level of significance
- $\boldsymbol{\beta}$iases
- $\boldsymbol{\zeta}$lock offsets
- $\boldsymbol{\delta t}$lock error [s]
- $\boldsymbol{\kappa}$orrection - relativity
- $\boldsymbol{\iota}$onosphere or $\boldsymbol{I}$
- $\boldsymbol{\tau}$roposphere or $\boldsymbol{T, T_h, T_w}$ [m]
- $\boldsymbol{M}$ elevation dependent mapping function for the troposphere wet delay
- $\boldsymbol{\xi}$ phase wind-up error
- $\boldsymbol{\epsilon}$rror in observations and unmodelled effects [m]

- $\boldsymbol{\phi_i^j}$ Carrier phase observable (times c) [m]
- $\boldsymbol{P_i^j}$ Pseudo range observable [m]

Lets try this for example: For an undifference, uncombined float solution, the linearized observation equations for pseudorange and phase observations from satellite s to receiver r can be described as:

$\Delta P_{r,f}^{q,s} = u_r^{q,s}.\Delta x + c.(\delta t_r^q - \delta t^{q,s}) + M_r^{q,s}.T_r + \gamma_f^q.I_{r,1}^{q,s} + d_{r,f}^q - d_f^{q,s} + \epsilon_{P,f}^q$

$\Delta\phi_{r,f}^{q,s} = u_r^{q,s}.\Delta x + c.(\delta t_r^q - \delta t^{q,s}) + M_r^{q,s}.T_r - \gamma_f^q.I_{r,1}^{q,s} + \lambda_f^q.N_{r,f}^{q,s} + b_{r,f}^q - b_f^{q,s} + \epsilon_{L,f}^q$

where$\Delta P_{r,f}^{q,s}$ and $\Delta\phi_{r,f}^{q,s}$ are the respective pseudorange and phase measurements on the frequency f(f=1,2), from which the computed values are removed; $u_r^{q,s}$ is the receiver-to-satellite unit vector; Δx is the vector of the receiver position corrections to its preliminary position; δt_r^q and $\delta t^{q,s}$ are the receiver and satellite clock errors respectively; c is the speed of light in a vaccum $M_r^{q,s}$ is the elevation dependent mapping function for the troposphere wet delay from the corresponding zenith one T_r; $I_{r,1}^{q,s}$ is the ionosphere delay along the line-of-sight from a receiver to a satellite at the first frequency and $\gamma_f^q = (\lambda_f^q/\lambda_1^q)^2$; λ_f^q is the wavelength for the frequency f of a GNSS q; $N_{r,f}^{q,s}$is the phase ambguity $d_{r,f}^q$ and $b_{r,f}^q$ are the receiver hardware delays of code and phase observations respectively; $d_f^{q,s}$ and $b_f^{q,s}$ are the satellite hardware delays of code and phase observations, respectively; $\epsilon_{P,f}$ and $\epsilon_{L,f}$ are the code and phase measurement noises respectively.

Flex Events

15.1 Introduction

Certain satellites in the GPS constellation have the ability to change the output power on the signals they transmit toward Earth. Effectively this is achieved by re-distributing the power allocated to each signal component as seen in Steigenberger et al. [2018a]. This means that individual signal components can therefore transmit above previously stated maximum [IS-GPS-200G, 2012, sec. 6.3.1].

This “flexible power” or “flex power” capability is used as an anti-jamming technique. When flex power is activated (or de-activated) by the Control Segment of the GPS system, it is sometimes referred to as a ”flex event”. When a series of flex events are associated with a given set of GPS satellites, alter the power spectral distribution in similar ways and/or are targeted over the same geographical region, this is categorised as a flex power mode [Steigenberger et al., 2018a]. In Steigenberger et al. [2018a], three flex power modes are discussed. This include a global activation for all healthy GPS Block IIR-M and IIF satellite for a period of 4 days (Mode II) to a geographically localised (regional) activation for 10 out of 12 Block IIF satellites on a continuous basis over a point centred in the Middle East (Mode I).

Since the launch of the first GPS block IIR-M satellite in 2005[1], every new GPS satellite has had programmable power output capabilities. This therefore includes the newer block satellites IIF and IIIA as well. Other constellations do not have this programmable variation, i.e. the power output is static.

GLONASS has greater variation in the total power output within a generation , i.e. different satellites in a given generation will vary by 10’s of watts. The power output of L1 and L2 frequencies will also differ with gradation of low/medium/high. This is covered in [Steigenberger et al., 2018a], but a good summary can be found in Steigenberger et al. [2019].

Galileo also has some variation in signal outputs across the In-Orbit Validation (IOV) satellites although the Full Operational Capability (FOC) satellites are designed to be constant [Bar-Sever, 2019]. According to the Steigenberger et al. [2018b] the power range for IOV sats is 95 - 134 W, and FOC sats is 254 - 273 W.

15.2 Ginan Code Example

In order to detect flex events, we require both RINEX3 observation files for a given station and time period, and the associated sp3 orbital files which detail the positions of the satellites. The Ginan code base includes Python scripts to automatically download and process files to find these flex events.

For example, running the following command:

```
python3 find_flex_events.py HOB200AUS 2021-02-15 2021-02-18 S2W /home/user/test/
    test_data/ -c_dir /home/user/test/test_csv/ -p -p_dir /home/user/test/test_plts/ -
    p_span 2400
```

will download RINEX3 and sp3 files for the `HOB200AUS` station between the dates `2021-02-15` and `2021-02-18` into the directory `/home/user/test/test_data/`, find any flex events (“Start” and “End”) for all GPS satellites, output a list of these events to a `csv` file in `/home/user/test/testcsv/` and output the results as a series of `.png` plots (one for each event).

Apart from the station, dates and download directory provided in the command, all others are optional;

- `-c_dir` specifies the directory to save the `csv` file;
- `-p` specifies that plots are to be produced;

[1] https://www.e-education.psu.edu/geog862/node/1773

- `-p_dir` specifies the directory to save the `png` plot files;
- `-p_span` specifies the time span for the plots (in seconds).

Part III

Using the software

Processing in Precise Point Positioning mode

Precise Point Positioning(PPP)

16.1 Pea PPP Processing examples

In this example we will process 24 hours of data from a permanent reference frame station. The algorithm that will be use an L1+L2 and L1+L5 ionosphere free combination. The log files and processing results can be found in /data/acs/pea/output/exs/EX01_IF/.

```
$ ./pea --config ../../config/Ex01-IF-PPP.yaml
$ grep "\$POS" /data/acs/pea/output/exs/EX01_IF/EX01_IF-ALIC.TRACE
```

And you should see the following:

```
$POS,2062,431940.000,0,-4052052.7956,4212836.0144,-2545104.6423,...
$POS,2062,431970.000,0,-4052052.7956,4212836.0144,-2545104.6423,...
```

16.1.1 Using the Ionosphere-free observable to process a static data set - the float solution

In this example we will process 24 hours of data from a permanent reference frame station. The algorithm will use an L1+L2 and L1+L5 ionosphere-free combination. The log files and processing results can be found in ‘¡path to pea¿/output/exs/EX01_IF/‘.

```
$ ./pea --config ../../config/EX01-IF-PPP.yaml
```

The pea will then have the following output in ¡path to pea¿/output/exs/EX01_IF/ :
EX01_IF20624.snx - contains the station position estimates in SINEX format EX01_IF-ALIC2019199900.TRACE - contains the logging information from the processing run

```
$ grep "REC_POS" /data/acs/pea/output/exs/EX01_IF/EX01_IF-ALIC201919900.TRACE >
    ALIC_201919900.PPP
```

This will pipe all of the receiver position results reported in the station trace file to a seperate file for plotting.

```
$ python3 /data/acs/pea/python/source/pppPlot.py --ppp /data/acs/pea/output/exs/EX01_IF/
    ALIC_201919900.PPP
```

This will then create the plots alic_pos.png, a time series of the difference between the estimated receiver position and the median estimated position. And the plot alic_snx_pos.png, a time series of the difference between the estimated receiver position and the IGS SINEX solution for Alic Springs on this day.

16.1.2 Single Frequency Processing

```
$ ./pea --config ../../config/Ex01-SF-PPP.yaml

$ grep "\$POS" /data/acs/pea/output/exs/EX01_SF/EX01_SF-ALBY202011500.TRACE
```

And you should see something similar to the following:

```
$POS,2062,431940.000,0,-4052052.7956,4212836.0144,-2545104.6423,0.00000043966020,...
$POS,2062,431970.000,0,-4052052.7956,4212836.0144,-2545104.6423,0.00000043965772,...
```

16.1.3 Processing realtime

To process a continuous GPS station in real-time you will need to access the data stream from a NTRIP stream and the correction products from a NTRIPCaster. Geoscience Australia is running a caster that

provides global data stream, a dense network of stream covering the Australian region, and correction procdust provide by IGS Analysis Centres. You will need to apply for an AUSCORS account, or use the new NTRIPCaster that streams using https.

You can apply for an account at the following link : New GA Account link

Once your have your account and password you can test that you can successfuly connect to the NTRIPCaster by using the following curl command:

```
$ curl https://ntrip.data.gnss.ga.gov.au/MOBS00AUS0 --http0.9 -i -u'<username>:<password
    > --output -'
```

16.2 Processing a Global Network to obtain satellite clock products

In this example 24 hours of data from a small global network of 87 stations is processed to obtain the clock products needed for high precision positioning.

Check that the paths in the configuration file for the products and RINEX files are correct for your system. If you have followed the convention layed out in the INSTALL.md document you should not need to amend anything.

To start the processing use the command:

```
$ ./pea --config ../../config/Ex02-Network.yaml
```

The process will take approximatelly 2-3 hours to complete depending on CPU performance. The log files and processing results can be found in /data/acs/pea/output/examples/Ex02, or the alternative directory you have specified in the configuration file.

Change into your output directory. You should find a .TRACE file for each station processed, and a PEA.SUM file.

```
$ cd /data/acs/pea/output/examples/Ex02
```

To verify your solution, first grep for the xp values:

```
$ grep 'network xp' netprocessing.out > xp_test.txt
```

and then run:

```
$ python3 /data/acs/pea/python/src/comprun.py --test /data/acs/output/xp_test.txt --
    standard /data/acs/pea/example/EX02/standard/xp_standard.txt
```

This will produce the plots Posdiff.png, recclk.png, satclk.png, and zwd.png.

16.3 Example 03 Processing a Global Network to obtain the orbit and clock products

In this example 24 hours of data from a small global network of 87 stations is processed to obtain the orbit and clock products needed for high precision positioning.

Check that the paths in the configuration file for the products and RINEX files are correct for your system. If you have followed the convention layed out in the INSTALL.md document you should not need to amend anything.

To start the processing use the command:

```
$ ./pea --config ../../config/Ex03-Network_Orbits.yaml
```

The process will take approximatelly 2-3 hours to complete depending on CPU performance. The log files and processing results can be found in /data/acs/pea/output/examples/Ex03, or the alternative directory you have specified in the configuration file.

Change into your output directory. You should find a .TRACE file for each station processed, and a PEA.SUM file.

```
$ cd /data/acs/pea/output/examples/Ex03
```

To verify your solution, first grep for the xp values:

```
$ grep 'network xp' netprocessing.out > xp_test.txt
```

and then run:

```
$ python3 /data/acs/pea/python/src/comprun.py --test /data/acs/output/xp_test --standard
    /data/acs/pea/example/EX03/standard/xp_standard.txt
```

This will produce the plots Posdiff.png, recclk.png, satclk.png, and zwd.png.

To compare the satellite clocks run:

```
$ python3 /data/acs/pea/python/src/compareclk.py --standard /data/acs/pea/example/EX03/
    standard/aus20624.clk  --test ./aus20624.clk
```

This will produce plots for the differences in satellite clocks G02 through to G32 as well as calculating the RMS and standard deviation with respect to the standard. G01.png does not exists at this has been used as the pivot satellite clock that we use to remove the bias from.

16.4 Example 04 Processing a Global Network to obtain Ionospheric Vertical Total Electron Content (VTEC) Maps

In this example 24 hours of data from a small global network of 87 stations is processed to obtain the IONEX formatted Ionosphere VTEC maps and SINEX formatted satellite Diferential Signal Biases (DSB). Ionospheric VTEC maps follows the IONEX 1.1 format, which can be found in: https://gssc.esa.int/wp-content/uploads/2018/07/ionex11.pdf The satellite bias follows the bias-SINEX format, which can be found in: http://ftp.aiub.unibe.ch/bcwg/format/draft/sinex_bias_100_dec07.pdf

Check that the paths in the configuration file for the products and RINEX files are correct for your system. If you have followed the convention layed out in the INSTALL.md document you should not need to amend anything.

To start the processing use the command:

```
$ ./pea --config <installation directory>/pea/config/Ex04-Ionosphere.yaml
```

The process will take approximatelly 1-2 hours to complete depending on CPU performance and setting specifications.

The IONEX and bias-SINEX files can then be used to obtain SPP and PPP positioning solutions as specified in PPPExamples.md

A utility to plot the TEC values in an IONEX file have can be found at:

¡installation directory¿/pea/python/source/plotIONEX.py This utility will ask for the path/filename of the IONEX file and a start and stop time in hhmmss format. The Python code will then generate a figure in IONEX_yyyy-mm-dd_hh:mm:ss.png format for each entry between the start and stop time.

16.5 Network Post-Processing - Ultra-rapid product example

We will run the pea in a post-processing mode, along side the POD, to produce orbits and clock in near-realtime, as would be run to produce an IGS ultra-rapid product

16.6 Network Real-time Processing

The PEA is designed to do some stuff.

ANTEX files can be downloaded from https://files.igs.org/pub/station/general/igs14.atx DCB files can be downloaded from ftp://ftp.aiub.unibe.ch/CODE/P1C1.DCB Satellite metadata can be obtained from: https://files.igs.org/pub/station/igs_satellite_metadata.snx

Part IV

Using the Source Code

Coding Standards

Coding Standards for C++

17.1 Code style

Decades of experience has shown that codebases that are built with concise, clean code have fewer issues and are easier to maintain. If submitting a pull request for a patch to the software, please ensure your code meets the following standards.

Overall we are aiming to

- Write for clarity
- Write for clarity
- Use short, descriptive variable names
- Use aliases to reduce clutter.

Unconcise code - Not recommended

```
//check first letter of satellite type against something

if (obs.Sat.id().c_str()[0]) == 'G')
    doSomething();
else if (obs.Sat.id().c_str()[0]) == 'R')
    doSomething();
else if (obs.Sat.id().c_str()[0]) == 'E')
    doSomething();
else if (obs.Sat.id().c_str()[0]) == 'I')
    doSomething();
```

Clear Code - Good

```
char& sysChar = obs.Sat.id().c_str()[0];

switch (sysChar)
{
    case 'G':   doSomething();   break;
    case 'R':   doSomething();   break;
    case 'E':   doSomething();   break;
    case 'I':   doSomething();   break;
}
```

17.2 Spacing, Indentation, and layout

- Use tabs, with tab spacing set to 4.
- Use space or tabs before and after any

  `+ - * / = < > == != % etc.`

- Use space, tab or new line after any , ;
- Use a new line after if statements.
- Use tabs to keep things tidy - If the same function is called multiple times with different parameters, the parameters should line up.

Scattered Parameters - Bad

```
trySetFromYaml(mongo_metadata,output_files,{"mongo_metadata" });
trySetFromYaml(mongo_output_measurements,output_files,{"mongo_output_measurements" });
trySetFromYaml(mongo_states,output_files,{"mongo_states" });
```

Aligned Parameters - Good

```
trySetFromYaml(mongo_metadata,              output_files, {"mongo_metadata"
    });
trySetFromYaml(mongo_output_measurements,   output_files, {"mongo_output_measurements"
    });
trySetFromYaml(mongo_states,                output_files, {"mongo_states"
    });
```

17.3 Statements

One statement per line - * unless you have a very good reason

Multiple Statements per Line - Bad

```
z[k]=ROUND(zb[k]); y=zb[k]-z[k]; step[k]=SGN(y);
```

Single Statement per Line - Good

```
z[k]    = ROUND(zb[k]);
y       = zb[k]-z[k];
step[k] = SGN(y);
```

Example of a good reason:

* Multiple statements per line sometimes shows repetetive code more clearly, but put some spaces so the separation is clear.

Normal

```
switch (sysChar)
{
    case ' ':
    case 'G':
        *sys = E_Sys::GPS;
        *tsys = TSYS_GPS;
        break;
    case 'R':
        *sys = E_Sys::GLO;
        *tsys = TSYS_UTC;
        break;
    case 'E':
        *sys = E_Sys::GAL;
        *tsys = TSYS_GAL;
        break;
    //...continues
}
```

Ok

```
if      (sys == SYS_GLO)    fact = EFACT_GLO;
else if (sys == SYS_CMP)    fact = EFACT_CMP;
else if (sys == SYS_GAL)    fact = EFACT_GAL;
else if (sys == SYS_SBS)    fact = EFACT_SBS;
else                        fact = EFACT_GPS;
```

Ok

```
switch (sysChar)
{
    case ' ':
    case 'G':   *sys = E_Sys::GPS;      *tsys = TSYS_GPS;    break;
```

```
    case 'R':   *sys = E_Sys::GLO;      *tsys = TSYS_UTC;    break;
    case 'E':   *sys = E_Sys::GAL;      *tsys = TSYS_GAL;    break;
    case 'S':   *sys = E_Sys::SBS;      *tsys = TSYS_GPS;    break;
    case 'J':   *sys = E_Sys::QZS;      *tsys = TSYS_QZS;    break;
//...continues
}
```

17.4 Braces

New line for braces.

```
if (pass)
{
    doSomething();
}
```

17.5 Comments

- Prefer '//' for comments within functions
- Use '/* */' only for temporary removal of blocks of code.
- Use '/** */' and '///¡' for automatic documentation

17.6 Conditional checks

- Put '&&' and '——' at the beginning of lines when using multiple conditionals.
- Always use curly braces when using multiple conditionals.

```
if  ( ( testA     > 10)
    &&( testB   == false
      ||testC   == false))
{
    //do something
}
```

* Use variables to name return values rather than using functions directly

Bad

```
if (doSomeParsing(someObject))
{
    //code contingent on parsing success? failure?
}
```

Good

```
bool fail = doSomeParsing(someObject);
if (fail)
{
    //This code is clearly a response to a failure
}
```

17.7 Variable declaration

- Declare variables as late as possible - at point of first use.
- One declaration per line.
- Declare loop counters in loops where possible.
- Always initialise variables at declaration.

```
int  type  = 0;
bool found = false;         //these have to be declared early so they can be used after the for loop

for (int i = 0; i < 10; i++)
{
    bool pass = someTestFunction();    //this pass variable isnt declared until it's used - good
    if (pass)
    {
        type  = typeMap[i];
        found = true;
        break;
    }
}

if (found)
{
    //...
}
```

17.8 Function parameters

- One per line.
- Add doxygen compatible documentation after parameters in the cpp file.
- Prefer references rather than pointers unless unavoidable.

```
void function(
        bool        runTests,           ///< Run unit test while processing
        MyStruct&   myStruct,           ///< Structure to modify
        OtherStr*   otherStr = nullptr) ///< Optional structure object to populate (cant use reference because its optional)
{
    //...
}
```

17.9 Naming and Structure

- For structs or classes, use 'CamelCase' with capital start
- For member variables, use 'camelCase' with lowercase start
- For config parameters, use 'lowercase_with_underscores'
- Use suffixes ('_ptr', '_arr', 'Map', 'List' etc.) to describe the type of container for complex types.
- Be sure to provide default values for member variables.
- Use heirarchical objects where applicable.

```
struct SubStruct
{
    int    type = 0;
    double val  = 0;
};

struct MyStruct
{
    bool          memberVariable = false;
    double        precision      = 0.1;

    double                      offset_arr[10]  = {};
    OtherStruct*                refStruct_ptr   = nullptr;

    map<string, double>         offsetMap;
```

```
    list<map<string, double>>  variationMapList;
    map<int, SubStruct>        subStructMap;
};

//...

MyStruct myStruct = {};

if (acsConfig.some_parameter)
{
    //..
}
```

17.10 Testing

- Use TestStack objects at top of each function that requires automatic unit testing.
- Use TestStack objects with descriptive strings in loops that wrap functions that require automatic unit testing.

```
void function()
{
    TestStack ts(__FUNCTION__);

    //...

    for (auto& obs : obsList)
    {
        TestStack ts(obs.Sat.id());

        //...
    }
}
```

17.11 Documentation

- Use doxygen style documentation for function and struct headers and parameters
- '/**' for headers.
- '///¡' for parameters

```
/** Struct to demonstrate documentation.
* The first line automatically gets parsed as a brief description, but more detailed
    descriptions are possible too.
*/
struct MyStruct
{
    bool    dummyBool;                  ///< The thing to the left is documented here
};

/** Function to demonstrate documentation
*/
void function(
        bool        runTests,           ///< Run unit test while processing
        MyStruct&   myStruct,           ///< Structure to modify
        OtherStr*   otherStr = nullptr) ///< Optional string to populate
{
    //...
}
```

17.12 STL Templates

- Prefer maps rather than fixed arrays.
- Prefer range-based loops rather than iterators or 'i' loops, unless unavoidable.

Bad

```
double double_arr[10] = {};

//..(Populate array)

for (int i = 0; i < 10; i++)    //Magic number 10 - bad.
{

}
```

```
map<string, double> doubleMap;

//..(Populate Map)

for (auto iter = doubleMap.begin(); iter != doubleMap.end(); iter++)   //long,
    undescriptive - bad
{
    if (iter->first == someVar)     //'first' is undescriptive - bad
    {
        //..
    }
}
```

Good - Iterating Maps

```
map<string, double> offsetMap;

//..(Populate Map)

for (auto& [siteName, offset] : doubleMap)  //give readable names to map keys and values
{
    if (siteName.empty() == false)
    {

    }
}
```

Good - Iterating Lists

```
list<Obs> obsList;

//..(Populate list)

for (auto& obs : obsList)         //give readable names to list elements
{
    doSomethingWithObs(obs);
}
```

Special Case - Deleting from maps/lists

Use iterators when you need to delete from STL containers:

```
for (auto it = someMap.begin(); it != someMap.end();  )
{
    KFKey key = it->first;              //give some alias to the key/value so they're
    readable

    if (measuredStates[key] == false)
    {
        it = someMap.erase(it);
    }
 else
 {
        ++it;
    }
}
```

17.13 Namespaces

Commonly used std containers may be included with 'using'

```cpp
#include <string>
#include <map>
#include <list>
#include <unordered_map>

using std::string;
using std::map;
using std::list
using std::unordered_map;
```

Python

18.1 Python Installation for Plotting, Processing, etc.

Lastly, to run many of the included scripts for fast parsing of .trace/.snx files, plotting of results, automatic running of the PEA based on input date/times and stations, etc. then a number of python dependencies are needed.

The file `python/ginan_py37.yaml` has a list of the necessary python dependencies.
The best way to take advantage of this is to install the Miniconda virtual environment manager.
This will allow you to pass the `.yaml` file into the `conda` command and automatically set up a new python environment.

To install miniconda, enter the following commands:

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash Miniconda3-latest-Linux-x86_64.sh
```

Listing 18.1: Installing conda

And follow the installation instructions on screen (choosing all defaults is fine).

Create virtual environment, after installation you can create the *ginan37* python environment. First open a new terminal session and enter:

```
$ conda env create -f <dir_to_pea>/python/ginan_py37.yaml
```

Listing 18.2: Example showing how to create a conda environment

You have now created the virtual python environment *ginan37* with all necessary dependencies. Anytime you wish you run python scripts, ensure you are in the virtual environment by activating:

```
$ conda activate gin37
```

Listing 18.3: Activating a conda environment

And then run your desired script.

18.2 Examples to run the PEA from a python script

The following examples will show how to use the python scripts in the `source` directory to automatically gather files and run the pea to produce results (as .TRACE files and plots).

18.3 Run the PEA

The easiest way to run the is to choose a reference station, a start time, end time and directories to store the gathered files and resultant outputs.
Doing this for station *HOB200AUS* and processing 2 days of data, starting 20 Dec 2020 00:00 and ending 21 Dec 2020 23:59:30, we can run the following code (assuming the `pwd` is your pea directory):

`python3 python/source/run_pea_PPP.py HOB200AUS 2020-12-20_00:00:00 2020-12-21_23:59:30 <directory_to_download_files_to> <results_output_directory> -md -del_yaml`

This will produce a .TRACE file that contains XYZ position information for that station and estaimes for the Zenith Total Delay (ZTD). The flags are as follows:
- `-md` is used to allow dates to be input in *YYYY-MM-DD* format as above. If the flag is not selected, the format must be *YYYY-DOY* (where *DOY* is day-of-year).
- `-del_yaml` is used to delete the `.yaml` config file after the pea run. Without this flag, the `.yaml` file will be kept (in `pwd`)

18.4 Near-Real-Time (NRT) Daily Run

In this example, we will see how to run the pea to produce ZTD and station coordinate results for reference station *HOB200AUS*, using the most rececnt data available on GA and CDDIS servers. Assuming the pwd is your `pea` directory, then the following code should run to produce a .TRACE file and plots of ZTD and XYZ station coordinates for the most recent day:

`python3 python/source/run_pea_PPP.py HOB200AUS _ _ <directory_to_download_files_to> <results_output -rapid -m_r -plt_m_r`

This differs to the code above in that the flags `-rapid`, `-m_r` and `plt_m_r` have been selected.
- `-rapid` will find the rapid versions of files - these are less accurate than final versions but are available sooner at CDDIS
- `-m_r` will get the code to search the CDDIS database and find the most recent rapid files available for `clk` and `sp3` files. The date and time is then selected based on this, allowing us to `_ _` for the start and end times
- `-plt_m_r` will plot the ZTD and XYZ results as `.png` files and place in a `plots` directory

18.5 Site metadata utility

Proper geodetic processing requires knowing of the site's receiver and antenna information. PEA extracts this information from the input sinex file or a set of sinex files. In addition, user can download igs.snx file that contains historical records of ∼500 sites and is regenerated daily from the logfiles available at ftp://ftp.igs.org/pub/station/log/. We developed a similar tool, log2snx.py, that does parsing of available igs logfiles and produces a sinex file as needed by PEA.

```
$ python3 scripts/log2snx.py  -h
  usage: log2snx.py [-h] [-l LOGGLOB] [-r RNXGLOB [RNXGLOB ...]] [-o OUTFILE]
  [-fs FRAME_SNX] [-fd FRAME_DIS] [-fp FRAME_PSD]
  [-d DATETIME] [-n NUM_THREADS]

IGS log files parsing utility. Globs over log files using LOGGLOB expression
and outputs SINEX metadata file. If provided with frame and frame
discontinuity files (soln), will project the selected stations present in the
frame to the datetime specified. How to get the logfiles: rclone sync
igs:pub/sitelogs/ /data/station_logs/station_logs_IGS -vv How to get the frame
files: rclone sync itrf:pub/itrf/itrf2014 /data/ITRF/itrf2014/ -vv --include
"*{gnss,IGS-TRF}*" --transfers=10 rclone sync igs:pub/ /data/TRF/ -vv
--include "{IGS14,IGb14,IGb08,IGS08}/*" see rclone config options inside this
script file Alternatively, use s3 bucket link to download all the files needed
s3://peanpod/aux/

optional arguments:
-h, --help            show this help message and exit
-l LOGGLOB, --logglob LOGGLOB
        logs glob path (required)
-r RNXGLOB [RNXGLOB ...], --rnxglob RNXGLOB [RNXGLOB ...]
        rinex glob path (optional)
-o OUTFILE, --outfile OUTFILE
        output file path (optional)
-fs FRAME_SNX, --frame_snx FRAME_SNX
        frame sinex file path (optional)
-fd FRAME_DIS, --frame_dis FRAME_DIS
        frame discontinuities file path (required with
        --frame_snx)
-fp FRAME_PSD, --frame_psd FRAME_PSD
        frame psd file path (optional)
-d DATETIME, --datetime DATETIME
        date to which project frame coordinates, default is
        today (optional)
-n NUM_THREADS, --num_threads NUM_THREADS
        number of threads to run in parallel
```

Listing 18.4: log2snx.py help message

18.6 Sinex preview utility

To quickly visualise and compare networks, a fast sinex preview utility has been developed - snx2map.py. It accepts any number of sinex files, extract estimated coordinates of the sites present and outputs a map in the form of html page.

```
$ python3 scripts/snx2map.py  -h
usage: snx2map.py [-h] [-i SNXPATH [SNXPATH ...]] [-o OUTDIR]

Parse sinex SITE/ID block and create html map.

optional arguments:
  -h, --help            show this help message and exit
  -i SNXPATH [SNXPATH ...], --snxpath SNXPATH [SNXPATH ...]
                        path to sinex file (.snx/.ssc). Can be compressed with
                        LZW (.Z)
  -o OUTDIR, --outdir OUTDIR
                        path to output dir
```

Listing 18.5: snx2map.py help message

Docker

Docker is "an open platform for developing, shipping, and running applications". It is a convenient way for us to provide all of the dependencies and the latest release source code so that we can use the ginan tool kit straight out of the box. In order for this to work, we will first need to install the docker engine onto our local machine. If we are running a different operating system instructions on how to install docker can be found at docker desktop downlod link, these also include alternative methods of installing on ubuntu and has links to recommended best practices. To find more information on docker have a look at the getting started guide provided by docker.

19.1 Ubuntu Docker dependency installation guide

If we are running ubuntu, we can install a docker engine. A summary of the commands to download and install docker involve setting up the ubuntu repository system to link with the docker repsotory are given below.

```
$ sudo apt -y update
$ sudo apt -y install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Add the dockers official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/
    share/keyrings/docker-archive-keyring.gpg
$ echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://
    download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/
    sources.list.d/docker.list > /dev/null
```

Then update the repository management system and install the packages.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Verify that the Docker engine is installed correctly by running the *hello-world* image.

```
$ docker run hello-world
```

Then we will need to include the current user to the *docker* group:

```
$ sudo usermod -aG docker root
$ sudo usermod -aG docker ${USER}
```

You will need to log out and back in for the permissions to take effect.

19.2 Using Docker

Once we have docker installed on our local machine we will need to download our image for Ginan:

```
$ docker pull gnssanalysis/ginan:latest
```

Then we can run the image as follows:

```
$ docker run -it -v /data:/data gnssanalysis/ginan:latest bash
```

This gives us a run-time environment where Ginan is installed with the executables *pea* and *pod* already in your path. Here, the *-v* option mounts a volume inside the docker instance at */data*, which maps to the */data* folder of the host machine. This way this folder can be shared between the host and the container, and the results can persist.

You should now see a *bash* prompt running inside the docker container. You can check the availability of Ginan executables by running

```
$ pea --help
$ pod --help
```

To run the standard testsuite, try:

```
$ /ginan/aws/docker/run-tests.sh
```

This docker image comes with a pre-built conda environment that enables plotting. To use it you have to activate the python conda environment first:

```
$ eval "$(/root/.miniconda3/bin/conda shell.bash hook)"
$ conda activate gn37
```

Finally, to exit this session, type:

```
$ exit
```

19.3 Keeping a container running

If we instantiate a container this way, our session will finish when we quit the *bash* prompt. The changes we make to the container will also be lost, except the changes that persist outside of the container, that is, the */data* folder in our example.

Therefore, it is sometimes useful to keep a container running, and connect to it and detach from it as needed.

To start up a docker container in the detached mode, run:

```
$ docker run -d -v /data:/data gnssanalysis/ginan:latest sleep infinity
```

we can verify that the container is running in the background:

```
$ docker ps
```

This will show a container ID. Docker conveniently also provides an alias as a “name“. We can start a new bash shell inside the container by:

```
$ docker exec -it <name> bash
```

where *¡name¿* is the name or ID of the running docker container.

Part V

YAML Configuration File Reference

PEA YAML Configuration

20.1 PEA processing options

20.2 Input File Options

```
input_files:

root_input_directory: /data/acs/pea/proc/exs/products

atxfiles:       [  igs14_2045_plus.atx                          ]  # Antenna models
snxfiles:       [  igs19P2062.snx                               ]  # meta data and apriori
    coords
blqfiles:       [  OLOAD_GO.BLQ                                 ]  # ocean loading is
    applied
navfiles:       [  brdm1990.19p                                 ]  # gnss broadcast file
sp3files:       [  gag20624.sp3                                 ]  # precise orbit data
erpfiles:       [  igs19P2062.erp                               ]  # earth orintation
    parameters
#dcbfiles:       [  CAS0MGXRAP_20191990000_01D_01D_DCB.BSX     ]  # monthly DCB file
#clkfiles:      [  jpl20624.clk                                 ]  # satellie and receiver
    clock
orbfiles:       [  gag20624_orbits_partials.out                 ]  # need this when
    estimating orbits (overrides .sp3 file)
```

Listing 20.1: yaml input files configuration example

20.3 RINEX station data

There are numerous ways that the *pea* can access GNSS RINEX observations to process. You can specify individual rinex files to process, set it up so that it will search a particular directory, or you can use a command line flag *-rnx ¡rnxfilename¿* to add an additional file to process. The data should be uncompressed (gunzipped, and not in hatanaka format), that is the *pea* expecting to just accept RINEX format].

```
station_data:

    root_stations_directory: /data/ginan/proc/data

    rnxfiles:
        - ALIC00AUS_R_20191990000_01D_30S_MO.rnx
```

Listing 20.2: pea yaml processing one station example

To process one RINEX file, you need to first specify the root dircetory of where the data is being stored in *root_stations_directory*, and then the name of the RINEX file as a single entry under *rnxfiles:*, as shown in the listing reflst:pea-yaml-single-station.

```
$ pea --rnx CAS100ATA_R_20191990000_01D_30S_MO.rnx
```

Listing 20.3: Example showing how to add a RINEX file to the processing list form the command line

If you wanted to process another file, located in the same *root_stations_directory*, this can be achieved at the command line using the *-rnx <rnxfilename >* flag, see listing reflst:pea-yaml-add-station.

```
station_data:

root_stations_directory: /data/acs/pea/proc/exs/data

```

```
rnxfiles:
- "*.rnx"                              #- searching all in file_root directory
```

Listing 20.4: yaml input files configuration example

20.4 Real-time streams

To process data in real-time you will need to set up the location, username annd password for the caster that you will be obtaining the input data streams from in the configuration file.

The pea supports obtaining streams from casters that use NTRIP 2.0 over http and https.

```
station_data:

    stream_root: "http://<usedname>:<password>@auscors.ga.gov.au:2101/"

    streams:
        - BCEP00BKG0
        - SSRA00CNE0
        - STR100AUS0
```

Listing 20.5: yaml input files configuration example

As shown in listing: 20.3, the caster url, username and password are specified within double quotes with the *stream_root* tag. In this example the streams are being obtained from the auscors caster run by Geoscience Australia. The broadcast information is being obtained from the stream *BCEP00BKG0* being supplied by BKG, and corrections to the utlra-rapid predicted orbit are being obtained from the stream *SSRA00CNE0*. The real-time data being processed is for the continuous GNSS station located at Mount Stromlo obtained from the stream *STR100AUS0*.

You can test your username and password is working correctly by running the curl command:

```
curl https://ntrip.data.gnss.ga.gov.au/ALIC00AUS0 -H "Ntrip-Version: NTRIP/2.0" -i  --
    output - -u <user>
```

20.5 output files:

```
output_files:

root_output_directory:          /data/acs/pea/output/<CONFIG>/

log_level:                      warn                            #debug, info, warn,
    error as defined in boost::log

output_trace:                   true
trace_level:                    2
trace_directory:                ./
trace_filename:                 <CONFIG>-<STATION><YYYY><DDD><HH>.TRACE

output_residuals:               true

output_persistance:             false
input_persistance:              false
persistance_directory:          ./
persistance_filename:           <CONFIG>.persist

output_config:                  false

output_summary:                 true
summary_directory:              ./
summary_filename:               PEA<YYYY><DDD><HH>.SUM

output_ionex:                   false
ionex_directory:                ./
ionex_filename:                 IONEX.ionex
iondsb_filename:                IONEX.iondcb

output_clocks:                  true
clocks_directory:               ./
clocks_filename:                <CONFIG>.clk
```

```
output_AR_clocks:                true

output_sinex:                    true
sinex_directory:                 ./
```

Listing 20.6: yaml input files configuration example

20.6 output options:

```
output_options:

config_description:              EX03_AR
analysis_agency:                 GAA
analysis_center:                 Geoscience Australia
analysis_program:                AUSACS
rinex_comment:                   AUSNETWORK1
```

Listing 20.7: yaml input files configuration example

20.7 processing Options

```
processing_options:

#start_epoch:               2019-07-18 00:00:00
#end_epoch:                 2017-03-29 23:59:30
#max_epochs:                300         #0 is infinite
epoch_interval:             30          #seconds

process_modes:
user:                   false
network:                true
minimum_constraints:    false
rts:                    false
ionosphere:             false
unit_tests:             false

process_sys:
gps:            true
#glo:           true
gal:            false
#bds:           true

elevation_mask:     10   #degrees

tide_solid:         true
tide_pole:          true
tide_otl:           true

phase_windup:       true
reject_eclipse:     true            #  reject observation during satellite eclipse
    periods
raim:               true
antexacs:           true

cycle_slip:
thres_slip: 0.05

max_inno:   0
max_gdop:   30

troposphere:
model:      gpt2    #vmf3
gpt2grid:   gpt_25.grd
#vmf3dir:    grid5/
#orography:  orography_ell_5x5

ionosphere:
corr_mode:      iono_free_linear_combo
iflc_freqs:     l1l2_only   #any l1l2_only l1l5_only

pivot_station:        "USN7"
#pivot_satellite:      "G01"
```

```
code_priorities: [  L1C, L1P, L1Y, L1W, L1M, L1N, L1S, L1L,
L2W, L2P, L2Y, L2C, L2M, L2N, L2D, L2S, L2L, L2X,
L5I, L5Q, L5X]
```

Listing 20.8: yaml input files configuration example

20.8 Network filter parameters

```
network_filter_parameters:

process_mode:               kalman      #lsq
inverter:                   LLT         #LLT LDLT INV

max_filter_iterations:      3
max_filter_removals:        3

rts_lag:                    -1      #-ve for full reverse, +ve for limited epochs
rts_directory:              ./
rts_filename:               <CONFIG>-Orbits.rts
```

Listing 20.9: yaml input files configuration example

20.9 Default filter parameters: stations

```
default_filter_parameters:

    stations:
        error_model:        elevation_dependent         #uniform elevation_dependent
            code_sigmas:        [0.30]
            phase_sigmas:       [0.003]
        pos:
            estimated:          false
            sigma:              [1.0]
            proc_noise:         [0]
            #apriori:                                   # taken from other source, rinex
     file etc.
            #frame:              xyz #ned
            #proc_noise_model:   Gaussian
            #clamp_max:          [+0.5]
            #clamp_min:          [-0.5]
        clk:
            estimated:          true
            sigma:              [0]
            proc_noise:         [1.8257418583505538]
            #proc_noise:         [10]
            #proc_noise_model:   Gaussian
        clk_rate:
            estimated:          false
            sigma:              [10]
            proc_noise:         [1e-4]
            #clamp_max:          [1000]
            #clamp_min:          [-1000]
        amb:
            estimated:          true
            sigma:              [100]
            proc_noise:         [0]
        trop:
            estimated:          true
            sigma:              [0.1]
            proc_noise:         [0.000083333]
        trop_grads:
            estimated:          false
            sigma:              [0.1]
            proc_noise:         [0.0000036]
```

Listing 20.10: yaml input files configuration example

20.10 Default filter parameters: satellites

```
satellites:

    clk:
        estimated:          true
        sigma:              [0]
        proc_noise:         [0.03651483716701108]

    clk_rate:
        estimated:          false
        sigma:              [0.01]
        proc_noise:         [1e-6]

    orb:
        estimated:          true
        sigma:              [5e-1, 5e-1, 5e-1, 5e-3, 5e-3, 5e-3, 5e-1]
```

Listing 20.11: yaml input files configuration example

20.11 Default filter parameters: eop

```
eop:
    estimated:  true
    sigma:      [30]
    #proc_noise: [0.0000036]
```

Listing 20.12: yaml input files configuration example

20.12 Ambiguity Resolution Options

```
ambiguity_resolution_options:
    Min_elev_for_AR:            15.0
    GPS_amb_resol:              true
    GAL_amb_resol:              false
    #Set_size_for_lambda:        10

WL_mode:                    iter_rnd        # AR mode for WL: off, round, iter_rnd, bootst, lambda, lambda_alt, lambda_al2, lambda_bie
    WL_succ_rate_thres:         0.9999
    WL_sol_ratio_thres:         3.0
    WL_procs_noise_sat:         0.00001
    WL_procs_noise_sta:         0.0001

NL_mode:                    iter_rnd        # AR mode for WL: off, round, iter_rnd, bootst, lambda, lambda_alt, lambda_al2, lambda_bie
    NL_succ_rate_thres:         0.9999
    NL_sol_ratio_thres:         3.0
    NL_proc_start:              86310

bias_read_mode:             30              # +1: read DSB biases, +2: read OSB biases, +4: read code biases, +8: read phase biases, +16: read satellite bias, +32: read station bias,
    bias_output_rate:           300.0
```

Listing 20.13: yaml ambiguity configuration example

PEA Configuration File - YAML

The PEA processing engine uses a single yaml file for configuration of all processing options.

21.1 YAML Syntax

The yaml format allows for heirarchical, self descriptive configurations of parameters, and has a straightforward syntax.

White-space (indentation) is used to specify heirarchies, with each level typically indented with 4 space characters.

Colons (:) are used to separate configuration keys from their values.

Lists may be created by either appending multiple values on a single line, wrapped in square brackets and separated by commas, or, by adding each value on a separate indented line with a dash before the value.

Adding a hash symbol (#) to a line will render the remainder of the line as a comment to be ignored by the parser.

Strings with special characters or spaces should be wrapped in quotation marks.

You will see all of these used in the example configuration files, but the files may be re-ordered, or re-formatted to suit your application.

21.2 Default Values

Many processing options have default values associated with them. To prevent repetition, and to ensure that the values are reported correctly, these values may be viewed in the acsConfig.hpp file within the source code directories.

21.3 input_files:

This section of the yaml file specifies the lists of files to be used for general metadata inputs, and inputs of external product data.

```
# Example
input_files:

    root_input_directory: /data/acs/pea/proc/exs/products/

    atxfiles:    [ igs14_2045_plus.atx                                  ]
    snxfiles:    [ igs19P2062.snx                                       ]
    blqfiles:    [ OLOAD_GO.BLQ                                         ]
    navfiles:    [ brdm1990.19p                                         ]
    orbfiles:    [ orb_partials/gag20624_orbits_partials_new.out        ]
    sp3files:    [ "*.sp3"                                              ]
    clkfiles:    [ jpl20624.clk                                         ]
    erpfiles:    [ igs19P2062.erp                                       ]
    dcbfiles:    [ CAS0MGXRAP_20191990000_01D_01D_DCB.BSX               ]
    bsxfiles:    []
    ionfiles:    []
```

Listing 21.1: A typical input_files section

21.3.1 Globbing

Files may be specified individually, as lists, or by searching available files using a globbed filename using the star character (*)

21.3.2 root_input_directory:

This specifies a root directory to be prepended to all other file paths specified in this section. For file paths that are absolute, (ie. starting with a /), this parameter is not applied.

21.3.3 atxfiles:

A list of ANTEX files to be used in processing. These may supply the antenna parameters to be used by satellites and receivers.

21.3.4 snxfiles:

A list of SINEX files to be used in processing. These may supply the initial positions and other metadata for receivers.

21.3.5 blqfiles:

A list of BLQ files to be used in processing. These may supply the ocean tide loading data.

21.3.6 navfiles:

A list of NAV files to be used in processing. These may supply the basic broadcast ephemerides for satellites.

21.3.7 orbfiles:

A list of ORB files to be used in processing. These may supply the PEA with orbital and ratiation pressure data from the Ginan's POD module, allowing precise orbit data to be passed between the two pieces of software.

21.3.8 sp3files:

A list of SP3 files to be used in processing. These may supply the ephemerides for higher precision processing.

21.3.9 clkfiles:

A list of CLK files to be used in processing. These may supply the clock offsets for satellites and receivers for higher precision processing.

21.3.10 erpfiles:

A list of ERP files to be used in processing. These may supply the earth rotation parameter information.

21.3.11 dcbfiles:

A list of DCB files to be used in processing. These may supply the differential code biases to assist with ambiguity resolution.

21.3.12 bsxfiles:

A list of BSINEX files to be used in processing. These may supply biases to assist with ambiguity resolution.

21.3.13 ionfiles:

A list of ION files to be used in processing. These may supply the ionospheric modelling parameters for single frequency processing.

21.4 output_files:

This section of the yaml file specifies options to enable outputs and specify file locations.

An example of this section follows:

```
output_files:

root_output_directory:          /data/acs/ginan/examples/<CONFIG>/

output_trace:                   true
trace_level:                    3
trace_directory:                ./
trace_filename:                 <CONFIG>-<STATION><LOGTIME>.TRACE

output_residuals:               false

output_config:                  true

output_summary:                 false
summary_directory:              ./
summary_filename:               <CONFIG>-<YYYY><DDD><HH>.SUM

output_clocks:                  true
clocks_directory:               ./
clocks_filename:                <CONFIG>.clk
```

Listing 21.2: output_files:

21.4.1 Wildcard Tags

Output filenames can include wildcards wrapped in ¡ ¿ brackets to allow more generic names to be used. While processing, these tags are replaced with details gathered from processing, and allows for automatic generation of, for example, hourly output files.

21.4.2 <CONFIG>

This is replaced with the 'config_description' value entered in the yaml file.

21.4.3 <STATION>

This is replaced with the 4 character station id of each station that generates a trace file.

21.4.4 <LOGTIME>

This is replaced with the (rounded) time of the epochs within the trace file.

If trace file rotation is configured for 1 hour, the ¡LOGTIME¿ wildcard will be rounded down to the closest hour, and subsequently change value once per hour and generate a separate output file for each hour of processing.

21.4.5 <DDD>, <D>, <WWWW>, <YYYY>, <YY>, <MM>, <DD>, <HH>

These are replaced with the components of time of the start epoch.

21.4.6 root_output_dir:

This specifies a root directory to be prepended to all other file paths specified in this section. For file paths that are absolute, (ie. starting with a /), this parameter is not applied.

21.4.7 [X]_directory:

Directory to output file [X] to, where [X] are the features below. May contain wildcard tags. May be relative to root_output_dir, or absolute. If the directory does not exist, it will be created.

trace_directory, summary_directory, clocks_directory, ionex_directory, biasSinex_directory, sinex_directory, persistance_directory

21.4.8 [X]_filename:

Filename to use for output of [X]. May contain wildcard tags. File will be created or overwritten if it already exists.

trace_filename, summary_filename, clocks_filename, ionex_filename, biasSinex_filename, sinex_filename, persistance_filename

21.4.9 trace_level:

Integer from 0-5 to specify verbosity of trace outputs. (5 - print everything)

21.4.10 trace_rotate_period, trace_rotate_period_units:

Granularity of length of time used for <LOGTIME>tags. These parameters may be used such that the filename of an output will change intermittently, and thus distribute the output over multiple files.

The <LOGTIME>tag is updated according to the epoch time, not the current clock time.

trace_rotate_period must be a numeric value, and trace_rotate_period_units may be one of seconds (default), minutes, hours, days, weeks, years, (with or without plural s).

21.4.11 output_residuals:

Boolean to print the residuals from kalman filter operation to relevant trace files.

21.4.12 output_config:

Boolean to print a copy of the yaml file to the top of each trace file. This may assist with keeping a record of the parameters used to generate the particular results contained in the file.

21.4.13 output_trace:

Boolean to generate per-station trace files.

21.4.14 output_summary:

Boolean to generate a network summary file.

21.4.15 output_clocks:

Boolean to generate RINEX formatted clock files from processed data.

21.4.16 output_AR_clocks:

Boolean to specify that the ambiguity resolved version of clocks should be output if output_clocks is enabled.

21.4.17 output_ionex:

Boolean to generate an IONEX file from processed ionosphere data.

21.4.18 output_ionstec:

Boolean to generate an IONSTEC file from processed ionosphere data.

21.4.19 output_biasSINEX:

Boolean to generate a biasSINEX from processed network data.

21.4.20 output_sinex:

Boolean to generate a sinex file containing processed solutions, and the metadata used to generate them.

21.4.21 output_persistance:

Boolean to save the network filter state, and navigation and ephemerides structure to disk once per epoch. For realtime processing where ephemerides are sourced from a a stream over several minutes, this may enable quicker start-up if the processor is restarted.

21.4.22 input_persisance:

Boolean to try to load a saved filter and navigation structure from disk.

21.4.23 output_mongo_measurements:

Boolean to output kalman filter measurements and residuals to a mongo database.

21.4.24 output_mongo_states:

Boolean to output the results of kalman filter processing to a mongo database.

21.4.25 output_mongo_logs:

Boolean to output timestamped log data from the console to a mongo database.

21.4.26 output_mongo_metadata:

Boolean to output timestamped metadata from processing to a mongo database. (unimplemented)

21.4.27 delete_mongo_history:

Boolean to delete a previous database using the same <CONFIG>tag before processing, to prevent collisions.

21.4.28 mongo_uri:

The URL to the location of the mongo database server.

21.5 station_data:

This section specifies the sources of observation data to be used in positioning.

It may consist of RINEX files, or RTCM streams, which are specified as follows:

Post processing:

```
# post processing example
station_data:
    root_stations_directory: /data/acs/ginan/examples/data
    rnxfiles:
        - "ALIC*.rnx"
        - "BAKO*.rnx"
```

Listing 21.3: station_data:

21.5.1 root_stations_directory:

This specifies a root directory to be prepended to all other file paths specified in this section. For file paths that are absolute, (ie. starting with a /), this parameter is not applied.

21.5.2 rnxfiles:

This is a list of RINEX files to be used for observation data. The first 4 characters of the filename are used as the receiver ID.

If multiple files are supplied with the same ID, they are all processed in sequence - according to the epoch times specified within the files. In this case, it is advisible to correctly specify the start_epoch for the filter, or the first epoch in the first file will likely be used.

21.5.3 rtcmfiles:

This is a list of RTCM binary files to be used for observation data. The first 4 characters of the filename are used as the receiver ID.

This can be used to read data that has been saved from a stream for later testing. There may be synchronisation issues when multiple such files are used.

Real-time processing:

```
# realtime streaming example
station_data:

    stream_root: "https://USERNAME:password@ntrip.data.gnss.ga.gov.au/"

    obs_streams:
    - BLCK00AUS0
    - BBDH00AUS0
    - BOMB00AUS0
    - BLGL00AUS0

    nav_streams:
    - SSR100AUS0
```

Listing 21.4: station_data:

21.5.4 stream_root:

This specifies a root url to be prepended to all other streams specified in this section. If the streams used have individually specified root urls, usernames, or passwords, this should not be used.

21.5.5 obs_streams:

This is a list of RTCM streams to read realtime data from. The first 4 characters of the filename are used as the receiver ID.

In combination with the stream_root parameter, they may require a username, password, port and mountpoint.

The streams in this section are processed for observations from receivers.

21.5.6 nav_streams:

This is a list of RTCM streams to read realtime data from.

In combination with the stream_root parameter, they may require a username, password, port and mountpoint.

The streams in this section are processed separately from observations, and will typically be used for receiving SSR messages or other navigational data from an external service.

21.6 processing_options:

This sections specifies the extent of processing that is performed by the engine.

21.6.1 epoch_interval:

Increment in nominal epoch time for each processing epoch. This parameter may be used to sub-sample datasets by using an epoch_interval that is a multiple of the dataset's internal interval between epochs.

21.6.2 start_epoch

Nominal time of the first epoch to process. Time is formatted as YYYY-MM-DD HH:MM:SS. This parameter may be left undefined to use the first available data point.

21.6.3 end_epoch

Maximum nominal time of the last epoch to process. This parameter may be left undefined.

21.6.4 max_epochs:

Maximum epochs to process before completion. This parameter may be left undefined.

21.6.5 process_modes:

```
process_modes:
    user:                   true
    network:                false
    minimum_constraints:    false
    rts:                    false
    ionosphere:             false
```

Listing 21.5: process_modes:

21.6.6 user:

Boolean to process all stations individually. Typically used for determining position of individual receivers. See TODO

21.6.7 network:

Boolean to process all stations in a single filter. May be used for determination of orbits, clocks, etc. See TODO

21.6.8 minimum_constraints:

Boolean to apply a rigid transformation to the results of the network filter after completion. See TODO

21.6.9 ionosphere:

Boolean to compute an ionosphere model from observations. See TODO

21.6.10 unit_tests:

Boolean to run tests to compare intermediate values during processing to stored results. See TODO

21.6.11 process_sys:

```
process_sys:
    gps:            true
    glo:            false
    gal:            false
    bds:            false
```

Listing 21.6: process_sys:

gps:

glo:

gal:

21.6.12 elevation_mask:

Minimum elevation required for observations to be used, measured in degrees.

21.6.13 ppp_ephemeris:

Option to specify source of satellite ephemeris used in PPP processing. Sources are
broadcast, precise, precise_com, sbas, ssr_apc, ssr_com

21.6.14 tide_solid:

Boolean to apply solid tide model to station positions.

21.6.15 tide_otl:

Boolean to apply ocean tide loading model to station positions.

21.6.16 tide_pole:

Boolean to apply pole tide model to station positions.

21.6.17 phase_windup

Boolean to apply phase windup model to satellite phase measurements.

21.6.18 reject_eclipse

Boolean to exclude eclipsed satellites from processing.

21.6.19 raim

Boolean to perform 'Receiver autonomous integrity monitoring' to detect and exclude observations that result in SPP failures.

21.6.20 cycle_slip:

thres_slip:

Threshold to apply to geometry free phase values to determine if an observation should be rejected due to a slip.

21.6.21 max_inno:

Maximum innovation in PPP measurement before both phase and code measurements are excluded.

21.6.22 deweight_factor:

Factor by which measurement variances are increased upon detection of a bad measurement.

21.6.23 max_gdop:

Maximum 'geometric dilution of precision' allowed for an SPP result to be valid.

21.6.24 antexacs:

Internal processing option. Bad things will likely happen if this is set to false.

21.6.25 sat_pcv:

Boolean to model satellite phase center variations.

21.6.26 pivot_station:

Station specified as origin for receiver clocks. Clocks for this station will be constrained to zero. May be set to <AUTO >or undefined to use first available station.

21.6.27 pivot_satellite

: Unused.

21.6.28 wait_next_epoch:

(under development) Expected time interval between successive epochs data arriving. For real-time this should be set equal to epoch_interval.

21.6.29 wait_all_stations:

(under development) Window of delay to allow observation data to be received for processing. Processing will begin at the earliest of: Observations received for all stations / wait_all_stations has elapsed since any station has received observations / wait_all_stations has elapsed since wait_next_epoch expired.

21.6.30 code_priorities:

List of observation codes that may be used in processing, and the order of priority for use. (Currently only a single code is used per frequency)

21.6.31 joseph_stabilisation:

Boolean to apply additional calculations in filter to ensure numerical stability.

21.7 user_filter_parameters, network_filter_parameters, ionosphere_filter_parameters:

21.7.1 inverter:

Method of inversion used to calculate kalman gain. options include LLT, LDLT, INV

21.7.2 max_prefit_remvovals:

Prior to kalman filtering the prefit residuals of measurements are checked for consistency, and may be removed or deweighted if not statistically reasonable. This parameter defines how many such measurements may be removed or deweighted at a given epoch.

These removals have low overhead associated with them as they occur before filter inversion takes place.

21.7.3 max_filter_iterations:

After kalman filtering, the postfit residuals of meaurements are checked for consistency, and measurements may be removed or deweighted if not statistically reasonable. This parameter specifies the number of times the filter may be iterated by deweighting and recomputing the resultant values.

These iterations have large overhead with them as the complete filter stage is computed multiple times. Large iteration values may exceed double the processing time.

21.7.4 rts_lag:

Number of future epochs to use in RTS smoothing. A larger lag will give more optimal smoothing results, at the expense of a longer lag before they are calculated, and requiring more processing time per epoch.

A negative value indicates that the entire solution should be smoothed at the conclusion of processing. This will obtain optimal results, with lowest processing time, but is not suitable for real-time applications.

21.7.5 rts_directory:

Directory to output RTS files.

21.7.6 rts_filename:

Filename for RTS files. Multiple intermediate files are generated by RTS smoothing, as well as an additional output files for kalman filter states and clocks.

21.7.7 outage_reset_limit:

Maximum number of epochs with missed phase measurements before the ambiguity associated with the measurement is reset.

21.7.8 phase_reject_limit:

Maximum number of phase measurements to reject before the ambiguity associated with the measurement is reset.

21.8 troposphere:

TODO aaron, in config twice

satellite options
Kalman objects
eigen
minimum constraint options

POD YAML Configuration

The YAML configuration file for the POD allows you to specify how the and what data the POD will process and what results and statistics to report at the end. In order to use the yaml configuration file you will need to specify this at the command line, with the *-y ¡yaml_filename¿* otherwise it will default to the traditional *POD.in, EQM.in and VEQ.in* option files.

```
$ pod -y example_configuration.yaml
```

Listing 22.1: calling the yaml configuration file

22.1 POD processing options (pod_options)

These options will control how the pod will process the input files, with four different options available. Only one of the options listed below can be set to true, the remainder must be set to false.

Option	Values	Comments
pod_mode_fit	true or false	Orbit Determination (pseudo-observations; orbit fitting)
pod_mode_predict	true or false	Orbit Determination and Prediction
pod_mode_eqm_int	true or false	Orbit Integration (Equation of Motion only)
pod_mode_ic_int	true or false	Orbit Integration and Partials (Equation of Motion and Variational Equations) initial condition integration

Table 22.1: POD YAML: processing options

```
pod_options:
# Example YAML showing different processing options
#-------------------------------------------------
pod_mode_fit:     true
pod_mode_predict: false
pod_mode_eqm_int: false
pod_mode_ic_int:  false
```

Listing 22.2: pod_options yaml configuration example

1. **pod_mode_fit** - this is used to fit an existing sp3 file (this is sometimes referred to as pseudo observations) with the parameters that are set later on. See pod example 1
2. **pod_mode_predict** - determine an orbit from observations and then predict the orbits path
3. **pod_mode_eqm_int** - determine the equations of motion only
4. **pod_mode_ic_int** -set up the initial conditions

22.2 Time scale(time_scale)

Option	Values	Comments
TT_time	true or false	Terrestrial (TT)
UTC_time	true or false	Universal (UTC)
GPS_time	true or false	Satellite (GPS)
TAI_time	true or false	Atomic (TAI)

Table 22.2: POD YAML: Time scale options

```
    time_scale:
        TT_time:  false
        UTC_time: false
        GPS_time: true
        TAI_time: false
```

Listing 22.3: time_scale yaml configuration example

1. **TT_time** -
2. **UTC_time** -
3. **GPS_time** -
4. **TAI_time** -

22.3 Initial Conditions (IC)

22.3.1 IC input format (ic_input_filename)

one is true, the other is false. If icf selected the ic_filename value specifies the path to the file.

Option	Values	Comments
sp3	true or false	sp3 format file
icf	true or false	initial conditions file

Table 22.3: POD YAML: Initial Conditions input format options

```
    ic_input_format:
        sp3:  true   # Input a-priori orbit in sp3 format
        icf:  false  # Input a-priori orbit in POD Initial Conditions File (ICF) format
        ic_filename: some_file
```

Listing 22.4: ic_input_format yaml configuration example

22.3.2 IC input reference system (ic_input_refsys)

reference system for the initial conditions one is true, the other is false.

Option	Values	Comments
itrf	true or false	terestrial
icrf	true or false	celestial
kepler	true or false	polar form of celestial

Table 22.4: POD YAML: Initial Conditions reference system

```
   ic_input_refsys:
        itrf: true   # Initial Conditions Reference Frame: ITRF, ICRF
        icrf: false  # Initial Conditions Reference Frame: ITRF, ICRF
        kepler: false
```

Listing 22.5: ic_input_refsys yaml configuration example

22.4 Using Pseudo observartions

These options are used to control how pseudo observations are used by the POD.

Option	Values	Comments
pseudobs_orbit_filename	filename	*path to the observations file*
pseudobs_interp_step	int	Interval (sec) of the interpolated orbit
pseudobs_interp_points	int	Number of data points used in Lagrange interpolation (at least 2)

Table 22.5: POD YAML: Using pseudo observations

```
    pseudobs_orbit_filename: igs19424.sp3 # Pseudo observations orbit filename
    pseudobs_interp_step:    900          # Interval (sec) of the interpolated orbit
    pseudobs_interp_points:  12           # Number of data points used in Lagrange
    interpolation
```

Listing 22.6: pseudo observation model yaml configuration example

22.5 Orbit arc length

Option	Values	Comments
orbit_arc_determination	int	*number of hours to integrate*
orbit_arc_prediction	int	*number of hours to predict at end of orbit arc*
orbit_arc_backwards	int	*number of hours to check before start of orbit arc*

Table 22.6: POD YAML: Orbit arc options

```
# Orbit arc length (in hours)
    orbit_arc_determination: 24  # Orbit Estimation arc
    orbit_arc_prediction:    12  # Orbit Prediction arc
    orbit_arc_backwards:     2   # Orbit Propagation backwards arc
```

Listing 22.7: orbit arc length yaml configuration example

22.6 External Orbit Comparison

In this section only one of the following options listed below can be set to true, the remainder must be set to false.

```

# External Orbit Comparison
    ext_orbit_enabled: true
    ext_orbit_type_sp3:       false        # Orbit data in sp3 format
                                           # (including position and velocity vectors)
    ext_orbit_type_interp:    true         # Interpolated orbit based on Lagrange
                                           # interpolation of sp3 file
    ext_orbit_type_kepler:    false        # Keplerian orbit
    ext_orbit_type_lagrange:  false        # 3-day Lagrange interpolation
    ext_orbit_type_position_sp3:    false  # Position and SP3 file
    ext_orbit_filename:       igs19424.sp3 # External (comparison) orbit filename
```

Option	Values	Comments
ext_orbit_enabled	true or false	
ext_orbit_type_sp3	true or false	
ext_orbit_type_interp	true or false	
ext_orbit_type_kepler	true or false	
ext_orbit_type_lagrange	true or false	
ext_orbit_type_position_sp3	true or false	
ext_orbit_filename	filename	*path to the orbit file*
ext_orbit_interp_step	int	Interval (sec) of the interpolatedKepler orbit
ext_orbit_interp_points	int	Number of data points used in Lagrange interpolation (at least 2)

Table 22.7: POD YAML: External orbit options

```
12    ext_orbit_interp_step:    900          # Interval (sec) of the interpolated/Kepler
      orbit
13    ext_orbit_interp_points:  12           # Number of data points used
14                                           # in Lagrange interpolation
```

Listing 22.8: orbit arc length yaml configuration example

22.6.1 External orbit reference frame (ext_orbit_frame)

Option	Values	Comments
itrf	true or false	terrestrial
icrf	true or false	celestial
kepler	true or false	kepler orbital elements

Table 22.8: POD YAML: External orbit reference system

```
1   ext_orbit_frame:
2        itrf: true            # External orbit reference frame - ITRF
3        icrf: false           # External orbit reference frame - ICRF
4        kepler: false
```

Listing 22.9: external orbit reference frame yaml configuration example

22.7 Earth Orientation Parameters

In this section only one of the following options listed below can be set to true, the remainder must be set to false.

22.7.1 EOP type

```
1    EOP_soln_c04:   true         # IERS C04 solution : EOP_sol = 1
2    EOP_soln_rapid: false        # IERS rapid service/prediction center (RS/PC) Daily :
     EOP_sol = 2
3    EOP_soln_igs:   false        # IGS ultra-rapid ERP + IERS RS/PC Daily (dX,dY) :
     EOP_sol = 3. Need both rapid_file AND igs_file
4    EOP_soln_c04_file: eopc04_14_IAU2000.62-now
5    EOP_soln_rapid_file: finals2000A.daily
6    ERP_soln_igs_file: igu18543_12.erp
7    EOP_soln_interp_points: 4    # EOP solution interpolation points
```

Listing 22.10: eop estimation options

22.7.2 IAU Precession-Nutation model

Option	Values	Comments
EOP_soln_c04	true or false	C04 is the IERS solution
EOP_soln_rapid	true or false	Rapid is the rapidprediction center solution
EOP_soln_igs	true or false	igs is the ultra-rapid solution using partials. To use this you need both the rapid file and partials file.
EOP_soln_c04_file	filename	
EOP_soln_rapid_file	filename	
ERP_soln_igs_file	filename	
EOP_soln_interp_points	int	

Table 22.9: POD YAML: Earth Orientation Parameter solution options

Option	Values	Comments
eop_soln_interp_points	int	number of data points to be used in an eop interpolation (at least 2!)
iau_model_2000	true or false	
iau_model_2006	true or false	

Table 22.10: POD YAML: EOP model options

```
    # IAU Precession-Nutation model:
    iau_model_2000: true         # IAU2000A: iau_pn_model = 2000
    iau_model_2006: false        # IAU2006/2000A: iau_pn_model = 2006

```

Listing 22.11: eop model

22.8 Input files

Option	Values	Comments
gravity_model_file	filename	
DE_fname_header	filename	Emphemeris header file
DE_fanme_data	filename	Emphemeris data file
ocean_tides_model_file	filename	
leapsec_filename	filename	leapseconds to be added
satsinex_filename	filename	sinex file with satellite meta-data

Table 22.11: POD YAML: Input files

```
# Gravity model file
gravity_model_file: goco05s.gfc
# goco05s.gfc, eigen-6s2.gfc, ITSG-Grace2014k.gfc

# Planetary/Lunar ephemeris - JPL DE Ephemeris
DE_fname_header: header.430_229
DE_fname_data:   ascp1950.430

# Ocean tide model file
ocean_tides_model_file: fes2004_Cnm-Snm.dat
# FES2004 ocean tide model

```

```
# Leap second filename
leapsec_filename: leap.second

# Satellite metadata SINEX
satsinex_filename: igs_metadata_2063.snx
```

Listing 22.12: yaml example for general input files

22.9 Output options

Option	Values	Comments
sp3_velocity	true or false	if you wish to write out the velocities for comparison
partials_velocity: true or false	if you wish to write velocity vector partials to the output file	

Table 22.12: POD YAML: Output options

```
# Write to sp3 orbit format: Option for write Satellite Velocity vector
sp3_velocity: false        #  Write Velocity vector to sp3 orbit

#--------------------------------------------------------------------
# Write partials of the velocity vector w.r.t. parameters into the orbits_partials
    output file:
partials_velocity: false   # Write out velocity vector partials wrt orbital state vector
     elements
```

Listing 22.13: yaml example for output file optionss

22.10 Variational Equation Options

Option	Values	Comments
veq_integration	true or false	pod mode overides it anyway. Ignore.
ITRF	true or false	reference_frame
ICRF	true or false	one must be true
kepler	true or false	

Table 22.13: POD YAML: VEQ ref system

```
# Variational Equations
veq_integration: false

#--------------------------------------------------------------------
# Reference System for Variational Equations'  - Partials & Parameter Estimation
veq_refsys:
itrs: true      # ITRS: Terrestrial Reference System
icrs: false     # ICRS: Celestial Reference System
kepler: false
```

Listing 22.14: yaml example for variational equation options

Option	Values	Comments
estimator_iterations	int	integrate this number of times, using the generated initial conditions from the previous run as a start point

Table 22.14: POD YAML: general options

22.11 General Options

```
    # Parameter Estimation
    estimator_iterations: 2
```

Listing 22.15: yaml example for output file options

22.12 Apriori solar radiation models

Option	Values	Comments
no_model	true or false	
cannon_ball_model	true or false	see Cannonball
simple_boxwing_model	true or false	
full_boxwing_model	true or false	

Table 22.15: POD YAML: Apriori SRP model

```
srp_apriori_model:
    no_model:    false
    cannon_ball_model:    true
    simple_boxwing_model:    false
    full_boxwing_model:    false
```

Listing 22.16: yaml example for apriori srp model options

22.12.1 Estimated Solar radiation models

Option	Values	Comments
ECOM1	true or false	
ECOM2	true or false	
hybrid	true or false	mix of ECOM1 and ECOM2
SBOXW	true or false	Simple box wing model
EMPirical models	true or false	Empirical is independent of the other four

Table 22.16: POD YAML: Estimated SRP models

```
srp_apriori_model:
    no_model:               false
    cannon_ball_model:      true
    simple_boxwing_model:   false
    full_boxwing_model:     false
```

Listing 22.17: yaml example for apriori srp model options

22.12.2 gravity_model

Type of gravity model to apply, only one option can be true.

Option	Values	Comments
central_force	true or false	
static_gravity_model	true or false	
time_variable_model	true or false	
iers_geopotential_model	true or false	

Table 22.17: POD YAML: Gravity Models

```
gravity_model:
    central_force:           false   # Central force gravity field             :
    gravity_model = 0
    static_gravity_model:    false   # Static global gravity field model       :
    gravity_model = 1
    time_variable_model:     true    # Time-variable global gravity field model :
    gravity_model = 2
    iers_geopotential_model: false   # IERS conventional geopotential model     :
    gravity_model = 3
```

Listing 22.18: yaml example for gravitational force model options

22.12.3 stochastic pulse (pulse)

Do not mix pulses in R/T/N (terrestrial) with pulses in (X/X/Z)

Option	Values	Comments
enabled	true or false	then if true:
epoch_number	int	number of epochs to apply pulses each day
offset	int	seconds until the first pulse of the day
interval	int	seconds between each pulse (after the first)
directions		
x_direction	true or false	
y_direction	true or false	
z_direction	true or false	
r_direction	true or false	
t_direction	true or false	
n_direction	true or false	

Table 22.18: POD YAML:stochastic pulse options

```
pulse:
enabled:      false
epoch_number:     1    # number of epochs to apply pulses
offset:       43200    # since the start of day
interval:     43200    # repeat every N seconds
directions:
    x_direction:   true
    y_direction:   true
    z_direction:   true
    r_direction:   false
    t_direction:   false
    n_direction:   false
reference_frame:
    icrf:          true
    orbital:       false
```

Listing 22.19: yaml example for gravitational force model options

22.13 EQM options

22.13.1 Integration Step

Option	Values	Comments
RK4_integrator_method	true or false	Do not use RK4 for veq as it is not implemented
RKN7_integrator_method	true or false	only one can be true
RK8_integrator_method	true or false	
integrator_step	int	step size in seconds

Table 22.19: POD YAML: Integration Step models

```
# Numerical integration method
# Runge-Kutta-Nystrom 7th order RKN7(6): RKN7, Runge-Kutta 4th order: RK4, Runge-Kutta 8
    th order RK8(7)13: RK8
integration_options:
    RK4_integrator_method: false
    RKN7_integrator_method: true
    RK8_integrator_method: false
    integrator_step: 900         # Integrator stepsize in seconds
```

Listing 22.20: yaml example for gravitational force model options

22.13.2 Gravity Field

Option	Values	Comments
enabled	true or false	and if true:
gravity_degree_max		maximum model terms in spherical harmonic expansion
timevar_degree_max		maximum time variable model terms in spherical harmonic expansion

Table 22.20: POD YAML: Gravity Models

```
# Gravitational Forces
gravity_field:
    enabled: true
    gravity_degree_max:      15      # Gravity model maximum degree/order (d/o)
    timevar_degree_max:      15      # Time-variable coefficients maximum d/o
```

Listing 22.21: yaml example for gravitational force model options

22.13.3 planetary_perturbations:

Option	Values	Comments
enabled	true or false	Uses the emphemeris

Table 22.21: POD YAML: planetary perturbations

```
# Planetary Gravitational Forces
    planetary_perturbations:
        enabled: true
```

Listing 22.22: yaml example for planetary pertubations

22.13.4 tidal_effects:

```
    tidal_effects:
    enabled: true
    solid_tides_nonfreq:    true   # Solid Earth Tides frequency-independent terms
```

Option	Values	Comments
solid_tides_nonfreq	True or False	frequency independent Solid Earth Tides
solid_tides_freq	True or False	frequency dependent Solid Earth Tides
ocean_tides	True or False	uses the ocean tides file
solid_earth_pole_tides	True or False	tide induced earth spin rotation not about the centre of the ellipsoid
ocean_pole_tide	True or False	ocean response to the above
ocean_tides_degree_max	True or False	maximum model term in spherical harmonic expansion

Table 22.22: POD YAML: tidal effects

```
    solid_tides_freq:       true   # Solid Earth Tides frequency-dependent terms
    ocean_tides:            true   # Ocean Tides
    solid_earth_pole_tides: true   # Solid Earth Pole Tide
    ocean_pole_tide:        true   # Ocean Pole Tide
    ocean_tides_degree_max: 15     # Ocean Tides model maximum degree/order
```

Listing 22.23: yaml example for tidal effects

22.14 relativistic_effects:

Option	Values	Comments
enabled	true or false	Lens Thinning, SchwarzChild and deSitter effects, there are no means to separate these effects currently. The Lens Thirring effect is calculated but subsequently ignored in the POD.

Table 22.23: POD YAML:relativistic_effects

22.15 non_gravitational_effects:

22.15.1 Models to be applied:

Option	Values	Comments
solar_radiation	true or false	radiation push from the sun
earth_radiation	true or false	radiation push from the earth
antenna_thrust	true or false	reverse thrust from antenna radiation

Table 22.24: POD YAML: non gravitational effects

```
# Non-gravitational Effects
non_gravitational_effects:
    enabled: true
    solar_radiation: true
```

```
    earth_radiation: true
    antenna_thrust:  true
```

Listing 22.24: yaml example for non gravitational effects

22.15.2 Empirical parameters

Option	Values	Comments
ecom_d_bias	true or false	
ecom_y_bias	true or false	
ecom_b_bias	true or false	
ecom_d_cpr	true or false	(only ECOM1hybrid)
ecom_y_cpr	true or false	(only ECOM1hybrid)
ecom_b_cpr	true or false	
ecom_d_2_cpr	true or false	(only ECOM2hybrid)
ecom_d_4_cpr	true or false	(only ECOM2hybrid)
emp_r_bias	true or false	
emp_t_bias	true or false	
emp_n_bias	true or false	
emp_r_cpr	true or false	
emp_t_cpr	true or false	
emp_n_cpr	true or false	
cpr_count	int	empirical cpr count

Table 22.25: POD YAML: Configuration options for solar radiation pressure models

```
# Non-gravitational Effects
srp_parameters:
    ECOM_D_bias:  true
    ECOM_Y_bias:  true
    ECOM_B_bias:  true
    EMP_R_bias:   true
    EMP_T_bias:   true
    EMP_N_bias:   true
    ECOM_D_cpr:   true
    ECOM_Y_cpr:   true
    ECOM_B_cpr:   true
    ECOM_D_2_cpr: false
    ECOM_D_4_cpr: false
    EMP_R_cpr:    true
    EMP_T_cpr:    true
    EMP_N_cpr:    true
    cpr_count:    1
```

Listing 22.25: yaml example for srp parameters

NB EQM and VEQ srp parameters MUST be identical. May move into pod_options in future. overrides are not implemented yet. Ignore for now. We imagine overrides at the system, block (sat type) and individual satellite level

22.16 overides*

***This section has not yet been implemented in the POD, and is a placehodler for future versions.**

In this section put any system, block or PRN overrides that are different to the ones chosen before

```
overrides:
    system:
    GPS:
    srp_apriori_model:
    no_model:   false
    cannon_ball_model:   true
    simple_boxwing_model:   false
    full_boxwing_model:     false
```

```
    GAL:
    srp_apriori_model:
    no_model:   false
    cannon_ball_model:   false
    simple_boxwing_model:   false
    full_boxwing_model:     true
    GLO:
    srp_apriori_model:
    no_model:   false
    cannon_ball_model:   false
    simple_boxwing_model:   true
    full_boxwing_model:     false
    BDS:
    srp_apriori_model:
    no_model:   false
    cannon_ball_model:   false
    simple_boxwing_model:   true
    full_boxwing_model:     false
block:
    GPS-IIF:
        srp_apriori_model:
        no_model:   false
        cannon_ball_model:   false
        simple_boxwing_model:   true
        full_boxwing_model:     false
    # GPS BLK IIF use ECOM2 parameters
    srp_parameters:
        ECOM_D_bias:   true
        ECOM_Y_bias:   true
        ECOM_B_bias:   true
        ECOM_D_2_cpr: false
        ECOM_D_4_cpr: false
        ECOM_B_cpr:   true
prn:
    G01:
        srp_apriori_model:
            no_model:   false
            cannon_ball_model:   false
            simple_boxwing_model:   false
            full_boxwing_model:     true
```

Listing 22.26: yaml example for override

Part VI

Backmatter

Manual conventions and Tips

Converting latex to markdown pandoc

To display code listings use the package https://www.overleaf.com/learn/latex/Code_listinglistings.

Acknowledgements

We wish to acknowledge the use of and heritage of some of the source code that we have used to help develop the Ginan.

24.0.1 Eclipse Routine

The routines to calculate the eclipsing times for GPS satellites were based off the original routines written by Jan Kouba, they have since been heavily modified.

24.0.2 JPL Planetary Ephemerides

We are using the Jet Propulison (JPL) Planetary and Lunar Ephemerides processing program (ftp://ssd.jpl.nasa.gov/pub/eph/planets/fortran/), in particular the routines: `CONST.f, FSIZER3.f, INTERP.f, PLEPH.f, SPLIT.f` We have modified the following subroutines: `asc2eph.f90, STATE.f90` so that there is no longer a dependency on a binary file produced in the original JPL form.

24.0.3 Standards of Fundamental Astronomy (SOFA) routines

We have used a number of routines obtained from SOFA, http://www.iausofa.org/: `anp.for, bi00.for, bpn2xy.for, bpn2xy.for, c2ixys.for, c2tcio.for, cal2jd.for, cp.for, cr.for, era00.for, fad03.for, fae03.for, faf03.for, faju03.for, fal03.for, falp03.for, fama03.for, fame03.for, fane03.for, faom03.for, fapa03.for, fasa03.for, faur03.for, fave03.for, gmst00.for, gmst06.for, gmst_iers.f03, ir.for, jd2cal.for, jdcalf.for, numat.for, nut00a.for, obl80.for, pn00a.for, pn00.for, pnm00a.for, pnm06a.for, pom00.for, pr00.for, rx.for, rxr.for, ry.for, rz.for, s00.for, s06.for, sp00.for, taiutc.for, tide_pole_oc.f90, tide_pole_se.f90, time_GPS.f90, time_TAI.f90, time_TT.f90, time_TT_sec.f90, time_UTC.f90, tr.for, xy06.for, xys00a.for, xys06a.for`

24.0.4 International Earth Rotation Service (IERS) routines

The following routines we originally sourced from the IERS: `interp_iers.f, CNMTX.F, FUNDARG.F, LAGINT.f, ORTHO_EOP.F, PMSDNUT2.F, RG_ZONT2.F, UTLIBR.F, IERS_CMP_2015.F`

Lost pages

This sections contains information in the hidden functionality that could be used for debugging. Below is an old POD help message, with all pre-yaml options present:

```
Earth Radiation Model (ERM):   1

Default master POD config file = POD.in
To run from default config file: ../../bin/pod or ../../bin/pod -c POD.in

POD.in config file options by defaut can be overridden on the command line

Command line: ../../bin/pod -c -m -s -o -e -v -a -p -r -t -n -i -u -q -k -w -y -h

Where:
    -c --config  = Config file name [Default POD.config]
    -m --podmode = POD Mode:
                                1 - Orbit Determination (pseudo-observations; orbit
    fitting)
                                2 - Orbit Determination and Prediction
                                3 - Orbit Integration (Equation of Motion only)
                                4 - Orbit Integration and Partials (Equation of Motion
    and Variational Equations)
    -s --pobs    = Pseudo observations orbit .sp3 file name
    -o --cobs    = Comparison orbit .sp3 file name
    -e --eqm     = EQuations of Motion input file name  [Default: EQM.in]
    -v --veq     = Variatinal EQuations input file name [Default: VEQ.in]
    -a --arclen  = Orbit Estimation Arc length (hours)
    -p --predlen = Orbit Prediction Arc length (hours)
    -r --eopf    = Earth Orientation Paramaeter (EOP) values file
    -t --eopsol  = Earth Orientation Paramaeter file type: (1,2,3)
                                1 - IERS C04 EOP
                                2 - IERS RS/PC Daily EOP
                                3 - IGS RP + IERS RS/PC Daily (dX,dY)
    -n --nutpre  = IAU Precession / Nutation model
                                2000 - IAU2000A
                                2006 - IAU2006/2000A
    -i --estiter = Orbit Estimatimation Iterations (1 or greater)
    -u --sp3vel  = Output .sp3 file with velocities
                                0 - Do not write Velocity vector to sp3 orbit
                                1 - Write Velocity vector to sp3 orbit
    -q --icmode  = Initial condition from parameter estimation procedure
    -k --srpmodel= 1: ECOM1, 2:ECOM2, 12:ECOM12, 3:SBOX
    -w --empmodel= 1: activated, 0: no estimation
    -d --verbosity = output verbosity level [Default: 0]
    -y --yaml = yaml config file
    -h --help.   = Print program help

Examples:

        ../../bin/pod -m 1 -q 1 -k 1 -w 0 -s igs16403.sp3 -o igs16403.sp3
        ../../bin/pod -m 2 -q 1 -k 1 -w 0 -s igs16403.sp3 -e EQMx.in -v VEQx.in -p 12

For orbit updates using Parameter Estimation Algorithm (PEA):
        ../../bin/pod -m 4 -q 2 -k 1 -w 0 -s igs16403.sp3 -o igs16403.sp3
```

Bibliography

Peter Steigenberger, Steffen Thoelert, and Oliver Montenbruck. Flex power on GPS Block IIR-M and IIF. *GPS Solutions*, 23(1):8, November 2018a. ISSN 1521-1886. doi: 10.1007/s10291-018-0797-8. URL https://doi.org/10.1007/s10291-018-0797-8.

IS-GPS-200G. Interface specification IS-GPS-200: Navstar GPS space segment/navigation user segment interfaces, 2012. URL https://www.gps.gov/technical/icwg/IS-GPS-200G.pdf.

Peter Steigenberger, Steffen Thoelert, and Oliver Montenbruck. GPS and GLONASS Satellite Transmit Power: Update for IGS repro3. page 5, 2019. URL http://acc.igs.org/repro3/TX_Power_20190711.pdf.

Yoaz Bar-Sever. Why Wait? Real-Time GNSS Monitoring for Infrastructure Protection and a Perspective on Galileo vs GPS, 2019. URL https://www.gps.gov/governance/advisory/meetings/2019-11/bar-sever.pdf.

Peter Steigenberger, Steffen Thoelert, and Oliver Montenbruck. GNSS satellite transmit power and its impact on orbit determination. *Journal of Geodesy*, 92(6):609–624, 2018b. ISSN 1432-1394. doi: 10.1007/s00190-017-1082-2. URL https://doi.org/10.1007/s00190-017-1082-2.

Listings