Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-test_sunlineSuKF

SUNLINE SWITCH-UKF MODULE AND TEST

Prepared by	T. Teil

Status: Initial document
Scope/Contents
This module implements and tests a Switch Unscented Kalman Filter in order to estimate the sunline direction.

Rev:	**Change Description**	**By**
Draft	Initial Revision	T. Teil

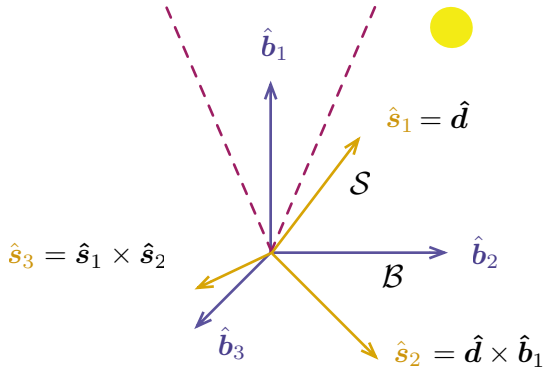

Fig. 1: Frame built off the body frame for Switch filters

Contents

---

1 Introduction

The Switch Unscented Kalman filter (SuKF) in the AVS Basilisk simulation is a sequential filter implemented to give the best estimate of the desired states. In this method we estimate the sun heading as well as the spacecraft rotation rate along the observable axes. The SuKF reads in the message written by the coarse sun sensor, and writes a message containing the sun estimate.

This document summarizes the content of the module, how to use it, and the test that was implemented for it. More information on the filter derivation can be found in Reference [**?**], and more information on the square root unscented filter can be found in Reference [**?**] (attached alongside this document).

2 Filter kinematics

2.1 Filter Derivation

The Switch-uKF attempts to avoid subtracting any terms from the state, while still removing the unobservable component of the rate. In order to do this, an appropriate frame must be defined. In order to not track the rate component alongside the sunline direction, that vector needs to be one of the basis vectors of the frame. It is decided to be the first vector for the frame, and therefore in that frame, ω_1 the component of the rotation rate can be removed from the states. This frame is called $\mathcal{S}_1 = \{\hat{\boldsymbol{s}}_1 = \frac{\boldsymbol{d}}{|\boldsymbol{d}|}, \hat{\boldsymbol{s}}_2, \hat{\boldsymbol{s}}_3\}$. This is seen in Figure **??**, where the dotted line represents the 30° threshold cone before switching frames.

The second vector of the frame must be created using only $\boldsymbol{d}$, and the body frame vectors. The first intuitive decision, is to use $\hat{\boldsymbol{b}}_1$ of the body frame and define $\boldsymbol{s}_2$ in Equation (**??**). The third vector $\hat{\boldsymbol{s}}_3$ of the $\mathcal{S}_1$ frame, is naturally created from the first two.

$$\hat{\boldsymbol{s}}_2 = \frac{\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_1}{|\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_1|} \qquad \hat{\boldsymbol{s}}_3 = \frac{\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{s}}_2}{|\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{s}}_2|} \tag{1}$$

The problem that arises is the singularity that occurs when $\hat{\boldsymbol{b}}_1$ and $\boldsymbol{d}$ become aligned: this frame becomes undefined. In order to counteract this, using a similar process as the shadow set used for Modified Rodrigues Parameters [**?**], a second frame is created. This frame $\mathcal{S}_2 = \{\hat{\bar{\boldsymbol{s}}}_1 = \hat{\boldsymbol{s}}_1, \hat{\bar{\boldsymbol{s}}}_2, \hat{\bar{\boldsymbol{s}}}_3\}$ is created with the same first vector, but constructs $\hat{\bar{\boldsymbol{s}}}_2$ using $\hat{\boldsymbol{b}}_2$ of the body frame as in Equation (**??**).

The last vector, once again, finishes the orthonormal frame.

$$\hat{\bar{\boldsymbol{s}}}_2 = \frac{\hat{\bar{\boldsymbol{s}}}_1 \times \hat{\boldsymbol{b}}_2}{|\hat{\bar{\boldsymbol{s}}}_1 \times \hat{\boldsymbol{b}}_2|} \tag{2}$$

With both these frames, $\mathcal{S}_1$ and $\mathcal{S}_2$, the singularities can always be avoided. Indeed, $\mathcal{S}_1$ becomes singular when $\boldsymbol{d}$ approches $\hat{\boldsymbol{b}}_1$, while $\mathcal{S}_2$ becomes singular when the sunheading approaches $\hat{\boldsymbol{b}}_2$. By changing frames, whenever the sunline gets within a safe cone of 30° (a modifiable value) of $\hat{\boldsymbol{b}}_1$, the frame is rotated into $\mathcal{S}_2$, which is not singular. Similarly, when $\boldsymbol{d}$ approches $\hat{\boldsymbol{b}}_2$ the frame is switched back to $\mathcal{S}_1$.

Because the two frames share the sunline vector $\boldsymbol{d}$, this vector is the same in both frames. This is a clear advantage as this is the vector we desire to estimate, and not having to rotate it avoids numerical issues. The rotation of the rates is done by computing the following DCMs, of which all the vectors are known.

$$[\mathcal{BS}_1] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_1 & {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_2 & {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_3 \end{bmatrix} \qquad [\mathcal{BS}_2] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_1 & {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_2 & {}^{\mathcal{B}}\hat{\bar{\boldsymbol{s}}}_3 \end{bmatrix} \qquad [\mathcal{S}_2\mathcal{S}_1] = [\mathcal{BS}_2]^T[\mathcal{BS}_1] \tag{3}$$

2.2 Filter Dynamics

The filter is therefore derived with the states being $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d} & \omega_2 & \omega_3 \end{bmatrix}^T$, given that $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = {}^{\mathcal{S}}\begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^T$. The rates of $\mathcal{S}$ relative to the body and inertial frame are related as such: $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} - \boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$. Since ω_1 is unknown, it is set to zero. Furthermore, since the sun heading is considered to be constant in the inertial frame over the period of time required for attitude determination and control, the equation becomes $-\bar{\boldsymbol{\omega}}_{\mathcal{S}/\mathcal{B}} = \bar{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$.

$\boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}}$ is estimated directly by the filter, and its skew matrix can be computed by setting ω_1 to zero (in the absence of information). This defines $\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$ as a function of known parameters. The dynamics are therefore given by Equations (**??**) and (**??**), where $[\tilde{\boldsymbol{d}}](2,3)$ corresponds to the 2^{nd} and 3^{rd} columns of the $[\tilde{\boldsymbol{d}}]$ matrix.

$$\boldsymbol{X}' = \boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d}' \\ \omega_2' \\ \omega_3' \end{bmatrix} = \begin{bmatrix} -{}^{\mathcal{B}}\bar{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{S}}\begin{bmatrix} 0 \\ \omega_2 \\ \omega_3 \end{bmatrix} \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

$$[A] = \left[\frac{\partial \boldsymbol{F}(\boldsymbol{d}, t_i)}{\partial \boldsymbol{X}}\right] = \begin{bmatrix} [{}^{\mathcal{B}}\tilde{\bar{\boldsymbol{\omega}}}_{\mathcal{S}/\mathcal{B}}] & -[\tilde{\boldsymbol{d}}][\mathcal{BS}](2,3) \\ [0]_{2\times 3} & [0]_{2\times 2} \end{bmatrix} \tag{5}$$

This formulation leads to simple dynamics, much simpler than those of the filter which subtracts the unobservable states, yet can actually estimate the observable of the rate, instead of using past estimates of $\boldsymbol{d}$.

In regard to the SR-uKF version of this filter, the same coefficients are used: $\alpha = 0.02$, and $\beta = 2$.

2.3 Switching Frames

When switching occurs, the switch matrix $[W]$ can be computed in Equation (**??**) using the previously computed DCMs. This equation assumes the switch is going from frame 1 to frame 2 (the reciprocal is equivalent), and $[\mathcal{S}_2\mathcal{S}_1](2,3)$ corresponds to the 2^{nd} and 3^{rd} columns of the $[\mathcal{S}_2\mathcal{S}_1]$ matrix.

$$[W] = \begin{bmatrix} [I]_{3\times 3} & [0]_{3\times 2} \\ [0]_{2\times 3} & [\mathcal{S}_2\mathcal{S}_1](2,3) \end{bmatrix} \tag{6}$$

The new states $\boldsymbol{X}$ and covariance [P] after the switch are therefore given in Equation (**??**)

$$\bar{\boldsymbol{X}} = [W]\boldsymbol{X} \qquad [\bar{P}] = [W][P][W]^T \tag{7}$$

When writing out the values of the state and covariance, it is necessary to bring it back into the body frame, using the $[\mathcal{BS}]$ DCM ($\mathcal{S}$ representing the current frame in use).

2.4 Measurements

The measurement model is given in equation **??**, and the H matrix defined as $H = \left[\frac{\partial \boldsymbol{G}(\boldsymbol{X},t_i)}{\partial \boldsymbol{X}}\right]^*$ is given in equation **??**.

In this filter, the only measurements used are from the coarse sun sensor. For the i^{th} sensor, the measurement is simply given by the dot product of the sunline heading and the normal to the sensor. This yields easy partial derivatives for the H matrix, which is a matrix formed of the rows of transposed normal vectors (only for those which received a measurement). Hence the H matrix has a changing size depending on the amount of measurements.

$$\boldsymbol{G}_i(\boldsymbol{X}) = \boldsymbol{n}_i \cdot \boldsymbol{d} \tag{8}$$

$$\boldsymbol{H}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{n}_1^T \\ \vdots \\ \boldsymbol{n}_i^T \end{bmatrix} \tag{9}$$

3 Filter Set-up, initialization, and I/O

3.1 User initialization

In order for the filter to run, the user must set a few parameters:

- The unscented filter has 3 parameters that need to be set, and are best as:
```
filterObject.alpha = 0.02
filterObject.beta = 2.0
filterObject.kappa = 0.0
```
- The angle threshold under which the coarse sun sensors do not read the measurement:
```
FilterContainer.sensorUseThresh = 0.
```
- The process noise matrix:
```
qNoiseIn = numpy.identity(5)
qNoiseIn[0:3, 0:3] = qNoiseIn[0:3, 0:3]*0.01*0.01
qNoiseIn[3:5, 3:5] = qNoiseIn[3:5, 3:5]*0.001*0.001
filterObject.qNoise = qNoiseIn.reshape(25).tolist()
```
- The measurement noise value, for instance:
```
FilterContainer.qObsVal = 0.001
```
- The initial covariance:
```
Filter.covar =
[1., 0.0, 0.0, 0.0, 0.0,
0.0, 1., 0.0, 0.0, 0.0,
0.0, 0.0, 1., 0.0, 0.0,
0.0, 0.0, 0.0, 0.02, 0.0,
0.0, 0.0, 0.0, 0.0, 0.02]
```

- The initial state :
  `Filter.state =[0.0, 0.0, 1.0, 0.0, 0.0]`

The messages must also be set as such:

- `filterObject.navStateOutMsgName = "sunline_state_estimate"`
- `filterObject.filtDataOutMsgName = "sunline_filter_data"`
- `filterObject.cssDataInMsgName = "css_sensors_data"`
- `filterObject.cssConfInMsgName = "css_config_data"`

3.2 Inputs and Outputs

The SuKF reads in the measurements from the coarse sun sensors. These are under the form of a list of cosine values. Knowing the normals to each of the sensors, we can therefore use them to estimate sun heading.

4 Test Design

The unit test for the sunlineSuKF module is located in:

`fswAlgorithms/attDetermination/sunlineSuKF/_UnitTest/test_SunlineSuKF.py`

As well as another python file containing plotting functions:
`fswAlgorithms/attDetermination/sunlineSuKF/_UnitTest/SunlineSuKF_test_utilities.py`

The test is split up into 3 subtests. The first test creaks up all of the individual filter methods and tests them individually. These notably go over the square-root unscented filter specific functions. The second test verifies that in the case where the state is zeroed out from the start of the simulation, it remains at zero. The third test verifies the behavior of the time update with a measurement modification in the middle of the run.

4.1 Individual tests

In each of these individual tests, random inputs are fed to the methods and their values are computed in parallel in python. These two values are then compared to assure that the correct computations are taking place.

- QR Decomposition: This tests the QR decomposition function which returns just the R matrix. Tolerance to absolute error $\epsilon = 10^{-15}$.

  Passed

- LU Decomposition: This tests the LU Decomposition accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- LU backsolve: This tests the LU Back-Solve accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- LU matrix inverse: This tests the LU Matrix Inverse accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- Cholesky decomposition: This tests the Cholesky Matrix Decomposition accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- L matrix inverse: This tests the L Matrix Inverse accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- U matrix inverse: This tests the U Matrix Inverse accuracy. Tolerance to absolute error $\epsilon = 10^{-12}$.

  Passed

4.2 Static Propagation

This test also takes no measurements in, and propagates with the expectation of no change. It then tests that the states and covariance are as expected throughout the time of simulation. Plotted results are seen in Figure **??**. We indeed see that the state and covariance that evolve nominally and without bias .

Tolerance to absolute error: $\epsilon = 10^{-10}$

4.3 Full Filter test

This test the filter working from start to finish. No measurements are taken in for the first 20 time steps. Then a heading is given through the CSS message. Halfway through the simulation, measurements stop, and 20 time steps later a different heading is read. The filter must be robust and detect this change. This test is parametrized for different test lengths, different initial conditions, different measured headings, and with or without measurement noise. All these are successful.

Tolerance to absolute error without measurement noise: $\epsilon = 10^{-10}$

Passed

Plotted results are seen in Figures **??**, and **??**. Figure **??** shows the state error and covariance over the run. We see the covariance initially grow, then come down quickly as measurements are used. It grows once again as the measurements stop before bringing the state error back to zero with a change in sun heading.

Figure **??** shows the post fit residuals for the filter, with no measurement noise. We see that the observations are read in well an that the residuals are brought back down to zero.

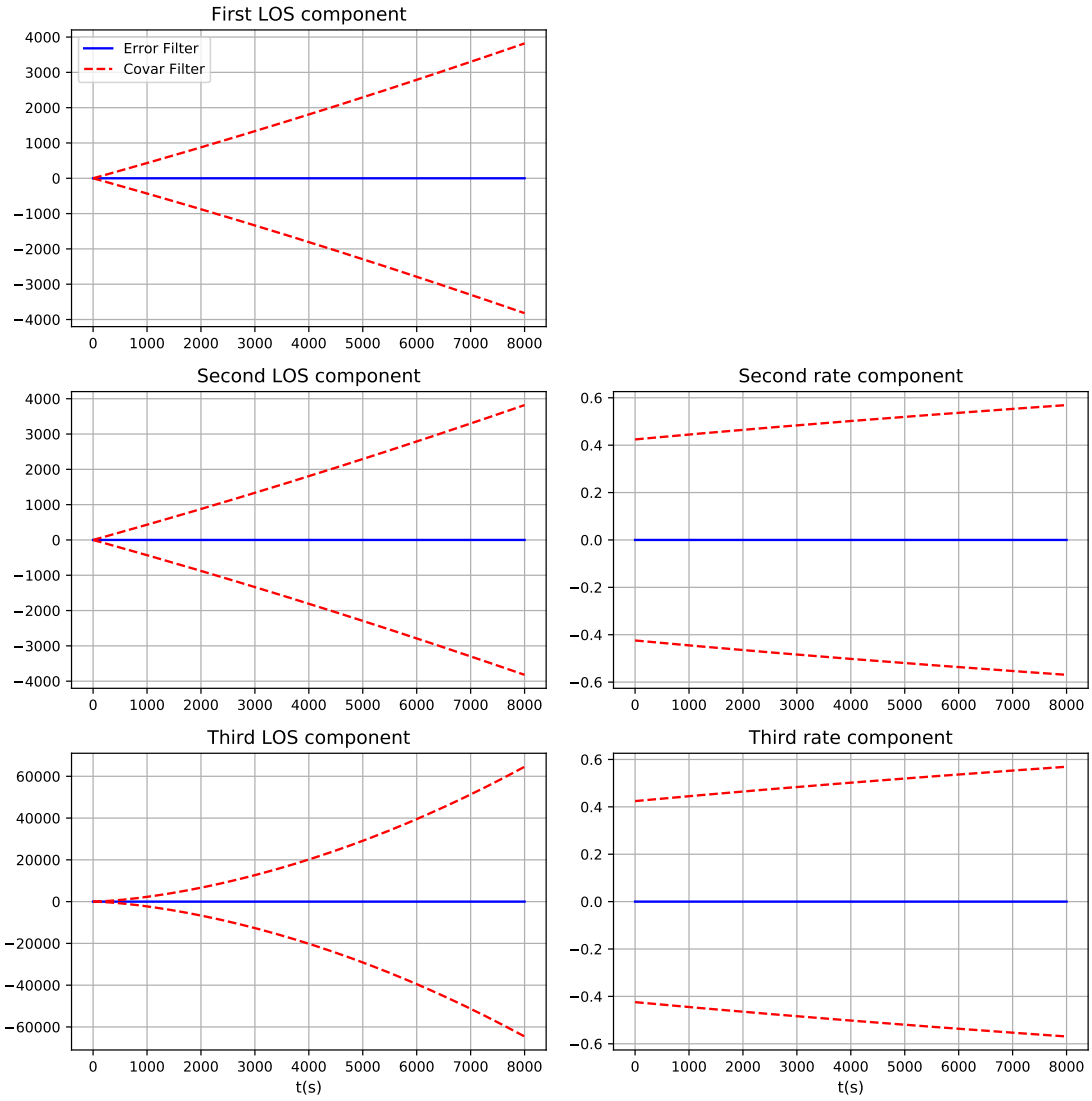

Fig. 2: State error and covariance

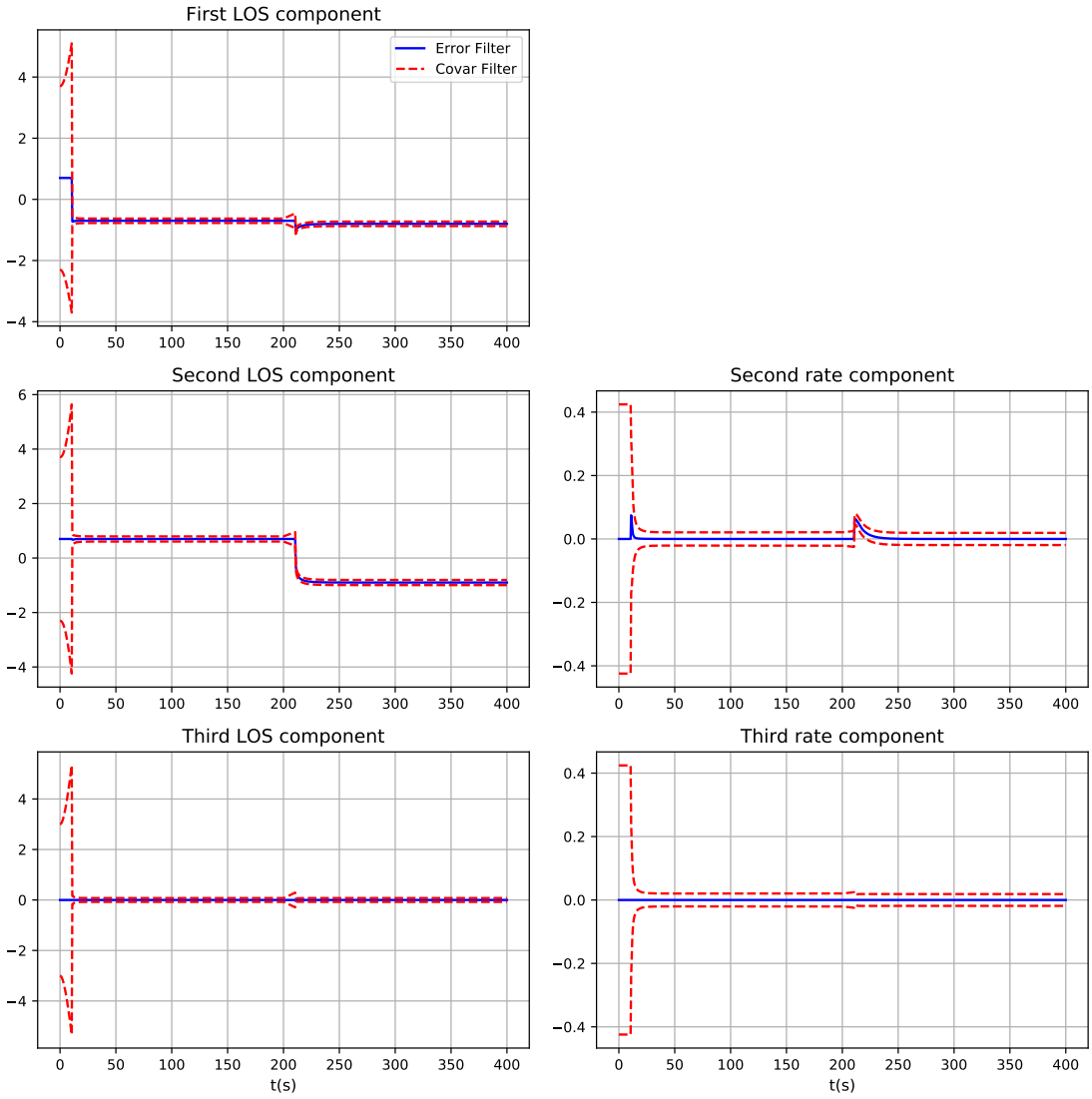

Fig. 3: State error and covariance

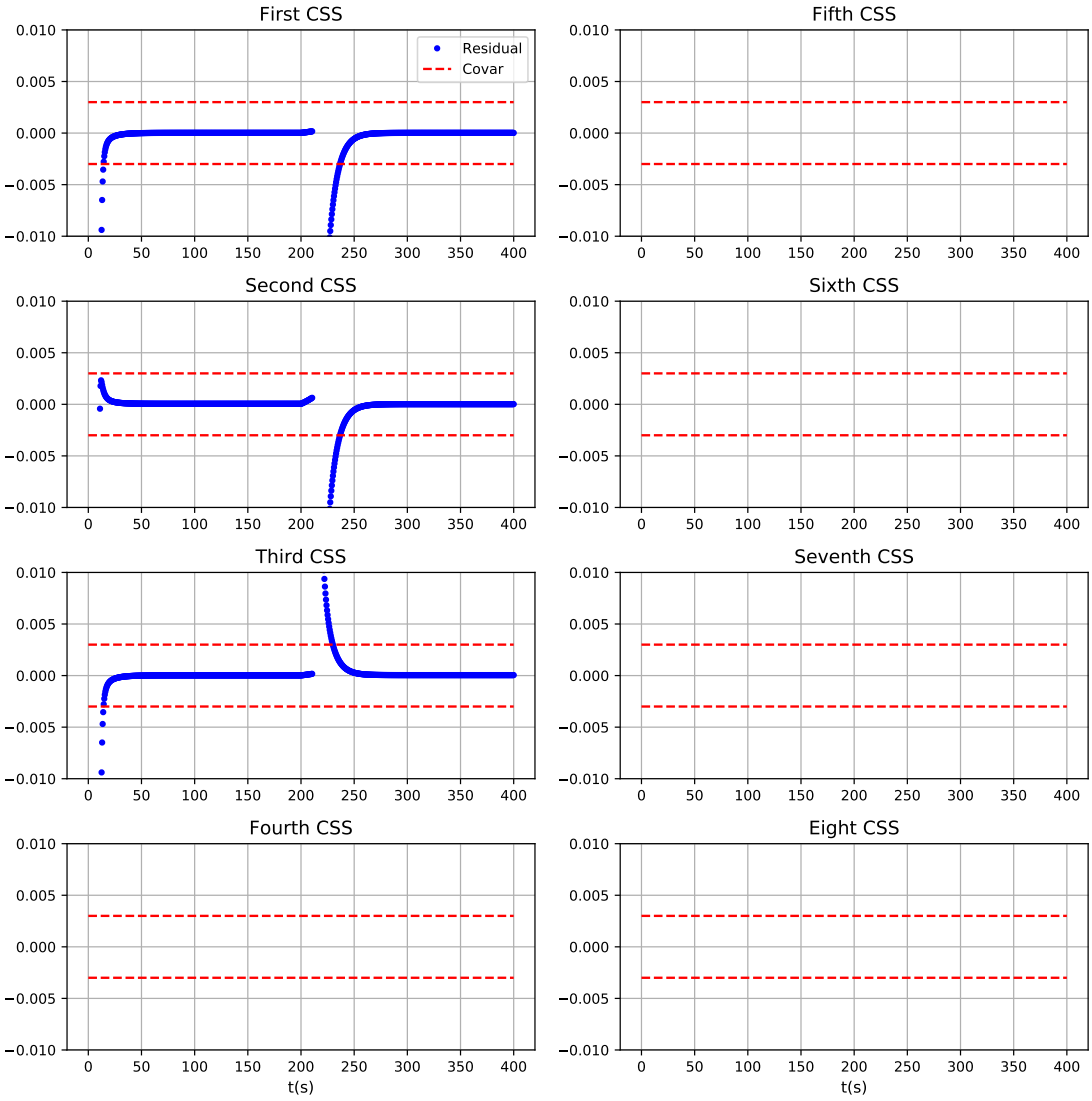

Fig. 4: Post Fit Residuals