Optimisation based observer

Federico Oliva, Daniele Carnevale

December 18, 2020

1 Algorithm rationale

This report presents the main procedure applied to implement the Optimisation based observer algorithm and some preliminary results.

1.1 Overview

This section describes the algorithm rationale. The observer aims to both estimate the state vector of the system considered and to identify model parameters. The steps are the following:

1. System definition: consider a general nonlinear system in state space form:

$$\begin{aligned} \dot{x} &= f(x, \theta, u) \\ y &= h(x, \theta) \end{aligned} \tag{1.1}$$

   where $x \in \mathbb{R}^n$ is the state vector, $\theta \in \Theta \subset \mathbb{R}^l$ the set of parameters to be identified, and $u \in \mathbb{R}^m$. Lastly, $y \in \mathbb{R}^p$ is the measurements vector. In order to include the parameters in the estimation process an augmented state is defined, resulting in a state space model with null dynamics in θ and the following state vector:

$$\xi = \begin{bmatrix} x_1 & \dots & x_n & \theta_1 & \dots & \theta_l \end{bmatrix}^T \in \mathbb{R}^{n+l} \tag{1.2}$$

2. Measurements down sampling: the measurements y are assumed to be collected every N_{Ts} sampling times and stored in a buffer of dimension w. The framework is described in Figure 1.

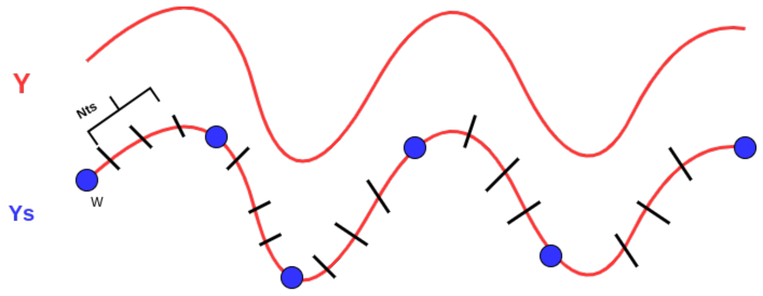


Figure 1: Signal measurements

   Therefore, every N_{Ts} time instants the measurements vector will be in the following form:

$$Y_i = \begin{bmatrix} y_i^1 \\ \vdots \\ y_i^p \end{bmatrix} \tag{1.3}$$

3. Optimisation procedure: the algorithm aims to find the initial state $\hat{\xi}_0$ best matching the measured values Y_i with $i \in \{1, \dots, w\}$. The general idea is presented in Figure 2.

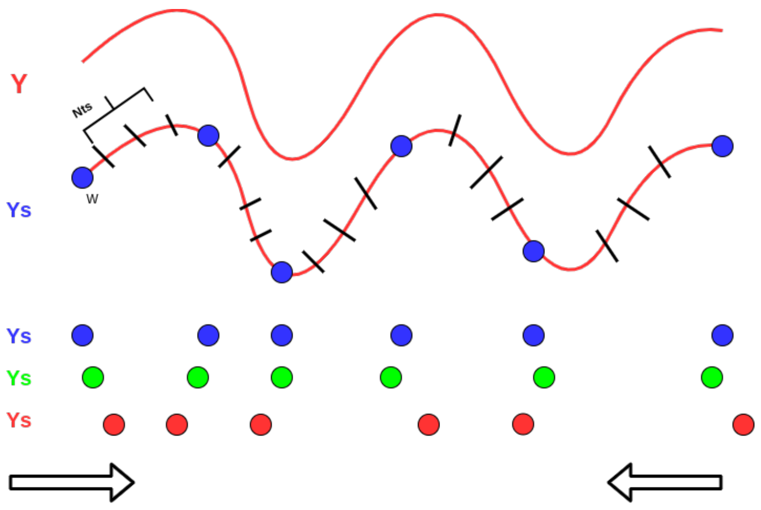


Figure 2: Signal optimisation: the arrows describe the possibility to run the code both forward and backward in time.

The optimisation is achieved by minimising a cost function J, depending on the estimation error, described as follows:

$$E_i = Y_i - H(\hat{\xi}_i) = \begin{bmatrix} y_i^1 - h_1(\xi_i) \\ \vdots \\ y_i^p - h_p(\xi_i) \end{bmatrix} \tag{1.4}$$
$$\mathbf{E} = \{E_i\},\ i \in \{1, \dots, w\}$$

The choice of such cost function will be discussed later on. Each value $\hat{\xi}_i$ is computed by propagating the initial condition $\hat{\xi}_0$; therefore, the algorithm can be used both forward and backward in time. In terms of numerical accuracy of the method, system stability shall be properly investigated before proceeding with the backward propagation[1].

1.2 Cost function and minimisation procedure

This section describes the cost function choice and the optimisation method. First the cost function is introduced and then the minimisation process is addressed.

[1]Check with Corrado

1.2.1 Cost function

The cost function is made by two terms, the former considers the measurements and the latter their time derivative. The general structure is the following:

$$J(Y, H(\hat{\xi})) = \sum_{i=1}^{w} \left(W_1^i (Y_i - \hat{Y}_i(\hat{\xi}))^2 + W_2^i (\dot{Y}_i - \hat{\dot{Y}}_i(\hat{\xi}))^2 \right) \tag{1.5}$$

where $Y_i, H(\hat{\xi}) \in \mathbb{R}^p$. The weights W_j^i are used to make the two terms in J of the same magnitude. The measurement derivative has been added to J in order to increase the information introduced in the observer. As an example, consider a mechanical system rotating around an axis and assume the inertia matrix to be unknown; the system is described by a double integrator and assume to measure the velocity of the system through a gyroscope. By differentiating the measure the acceleration is obtained. Clearly, the acceleration is strictly related to the rotational dynamics of the system. Therefore, by adding the time derivative of the gyroscope measures, the information about the system inertia is introduced in the estimation process. This concept along with a proper choice of both the input and the down sampling will play a key role in the observer tuning.

1.2.2 Minimisation - built in methods

Minimisation of J can be addressed in several ways; firstly the built-in MATLAB optimisation methods have been used. The following methods have been tested:

1. `fminunc`: unconstrained optimisation. This function uses gradient methods like the quasi-newton. The performance would be better if the gradient was provided analytically. This option has still to be developed.

2. `fmincon`: constrained optimisation. This function works similarly to the previous one but constraints are also set on the state components. The general performances are not better enough compared to the unconstrained to justify its use as default option. However, in some cases `fminunc` converges to values without a physical meaning. In those situations `fmincon` is a valid alternative.

3. `fminsearch`: this method exploits a simplex algorithm for nonlinear programming, avoiding the gradient computation. It's highly customizable and constraints can be added too. The performances are overall better than the other methods. However, by exploiting the analytical gradient the newton algorithms should perform the same or even better (see subsubsection 1.2.3).

Results will be presented later on, along with the tuning procedure[2].

1.2.3 Minimisation - sensitivity equations

Another approach proposed for the minimisation is based on the sensitivity function concept [2]. The main idea is to build and evaluate a function whose gradient returns an estimate of the variation in the system behaviour depending on both the state vector and parameters. This variation term will be used both to estimate the state and to identify the system. The steps are the following:

[2]For all the minimisation options see the code.

1. System definition: consider a general system in state space form, like the one described in Equation 1.1, namely

$$\begin{aligned}\dot{x} &= f(x, \theta, u)\\ y &= h(x, \theta)\end{aligned}$$

2. System flow: consider an initial condition x_0 and the solution of the system ODE, namely the flow $\phi(t, x_0)$. It holds

$$x(t) = \phi(t, x_0) \tag{1.6}$$

   This integration can be done both forward and backward in time.

3. Cost function: the cost function used as index for the optimisation procedure will be manipulated to extract an evaluation of the system variation on state and parameters. In order to explain the procedure the cost function defined in Equation 1.5 will be considered with $W_2^i = 0 \wedge W_1^i = 1 \ \forall i$, namely the measurements derivative won't be considered:

$$J(Y, H(\hat{x}_0)) = \sum_{i=1}^{w} \|y(t_i) - h(\phi(t_i, \hat{x}_0))\|^2 \tag{1.7}$$

4. Variation term: taking the derivative of J with respect to the initial condition the variation term is computed starting from a general point x_0:

$$\frac{\partial J}{\partial x_0} = -\sum_{i=1}^{w} (y(t_i) - h(\phi(t_i, x_0)))^T \frac{\partial h}{\partial x} \frac{\partial \phi}{\partial x_0}(t_i) \tag{1.8}$$

   This variation term can be rewritten as follows:

$$\frac{\partial J}{\partial x_0} = -\sum_{i=1}^{w} (y(t_i) - h(\xi_1(t_i)))^T \frac{\partial h}{\partial x} \xi_2(t_i) \tag{1.9}$$

   where the pair (ξ_1, ξ_2) is defined as a dynamic system as follows and integrated over time to construct the cost function:

$$\begin{aligned}\dot{\xi}_1 &= f(\xi_1), \ \xi_1 \in \mathbb{R}^n\\ \dot{\xi}_2 &= \frac{\partial f(\xi_1)}{\partial x} \xi_2, \ \xi_2 \in \mathbb{R}^{n \times n}\end{aligned} \tag{1.10}$$

5. Optimisation algorithm: once this gradient has been computed the optimisation procedure can be run on the initial condition x_0 through a gradient descent algorithm, namely iterating

$$x_0^{k+1} = x_0^k - \alpha \frac{\partial J}{\partial x_0} \tag{1.11}$$

   This procedure shall be performed any time a new measure is obtained, namely every N_{Ts} sampling times.

This approach is useful because it gives a glimpse on the physic behind the minimisation process. However, its main drawback consists in the need of an analytical expression for the gradient, which grows in complexity with the model.

1.3 Code workflow

This section briefly introduces the steps implemented in the code[3][4].

1. Init structure: run `Init_structure` to initialise the workspace. Next step will be referred to `ainOpt_DEZ_general_v14_fun_params`.
2. Init section and model selection (`line 1 - line 92`): in this section the main simulation settings are set. Moreover the system is initialised with an initial condition, the model function is set.
3. Reference trajectory definition (`line 96 - line 130`): the system is integrated in the defined time interval. This will be the reference trajectory of the system that the observer will try to reconstruct and identify.
4. Derivative setup (`line 132 - line 157`): define the variables for the numerical derivative computation.
5. Optimisation setup (`line 160 - line 270`): this section goes trough the corrupted initial condition generation, the optimisation setup, and the storage initialisation.
6. Observer implementation (`line 271 - line 433`): algorithm implementation:
   (a) Trajectory propagation (`line 277 - line 297`)
   (b) Measurements (`line 300 - line 356`): the measurement process shall be done at any time instant, in order to avoid errors in the derivative computation. The down sampling is exploited in the minimisation, namely only a restricted buffer of measures will be provided to the observer. In `line 303 - line 306` the sensors data measurement is modelled considering an additive noise.
   (c) Optimisation (`line 361 - line 412`): in this section the actual minimisation process is implemented. In case of forward propagation the optimisation initial state is shall be $N_{T_s} \times w$ time instants before the current one. Otherwise, the first time instant coincides with the current one. Depending on the optimisation setup either an unconstrained or constrained method is called in `line 375`. The cost function module change and the difference between the current and new state are computed. These entities could be used to decide whether consider the state innovation as useful or not. This is done in `line 392`. Currently `blue_flag = 1` meaning that every innovation is considered as good. Anyway, if the optimisation process is considered as good, the new state is propagated in order to update the trajectory.

[3]Main reference to `MainOpt_DEZ_general_v14_fun_params`

[4]**DISCLAIMER:** up to now the code is complete only for the forward optimisation

(d) Final measurements (`line 413 - 418`): after the new state is propagated, the measure array is updated with the new estimated value, as well as the derivative array. **Currently only the last values is updated, namely, the measure isn't updated for every instant back in time.**

7. Data analysis (`line 437 - line 475`): storage arrays are manipulated to ease the analysis and plot process.

Note that before any integration step the parameters are updated, both inside ans outside the optimisation process. If this isn't done, the identification won't work as the model function wouldn't propagate in a different way than the previous optimisation step.

2 Results and tuning

This section goes through some simulations run on different models. The goal is to test the algorithm on different situations and to draw some results based on performance evaluations.

2.1 Satellite model

The first test bench for the algorithm is a satellite. The goal of the test is to correctly identify the inertia matrix of the satellite, considering its rotational dynamics. In the following simulations the position of the satellite is considered as fixed, only the rotational motion is considered. Roughly speaking the model is the following:

$$\begin{aligned} \dot{q} &= \frac{1}{2}\Omega(\omega)q \\ \dot{\omega} &= -\frac{I}{\omega \wedge I R_{ECI}^{body}\frac{r}{\|r\|} + 3\mu \wedge I R_{ECI}^{body}\frac{r}{\|r\|} \cdot \frac{1}{r^3} + \tau} \end{aligned} \tag{2.1}$$

where $q \in \mathbb{R}^4$ is the quaternion describing the attitude and $\omega \in \mathbb{R}^3$ is the angular velocity. Therefore the state vector and the input are defined as

$$\begin{aligned} x &= \begin{bmatrix} q & \omega \end{bmatrix}^T \in \mathbb{R}^7 \\ u &= \tau \in \mathbb{R}^3 \end{aligned} \tag{2.2}$$

The system is provided with a gyroscope. Thus, the angular velocity is measured. The simulation setup is the following:

- Sampling time: $T_s = 1s$
- Simulation time: from 100 to 350s

The control τ is a simple PD controller[5].

[5]See code from SIA

2.2 Satellite - persistent excitation

The first simulation consider the system without any input. The setup is the following:

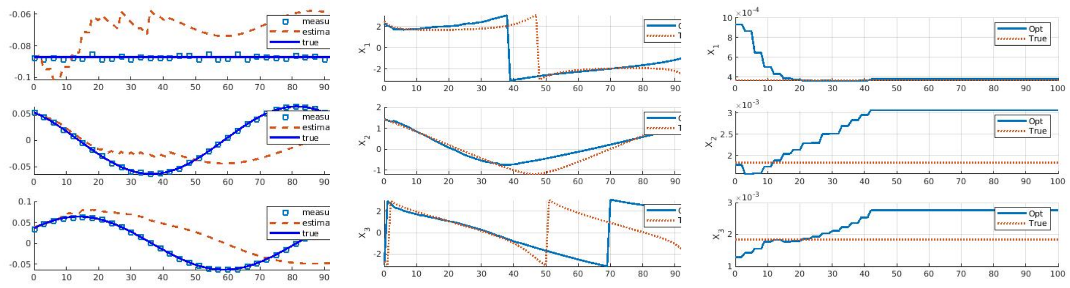
– down sampling:

$$w = 20, \tag{2.3}$$
$$N_{T_s} = 3 \tag{2.4}$$

– Control: no control action

– additive noise:
$$\delta_y = \mathcal{N}(0, 1e-3) \tag{2.5}$$

The results are presented in Figure 3. Clearly, the estimation isn't accurate and the main reason is the lack of information in the ω_x signal. The more the the signal is rich the more accurate the estimation will be.



(a) Windowed data (b) Euler angles (c) Parameters

Figure 3: Results - no input

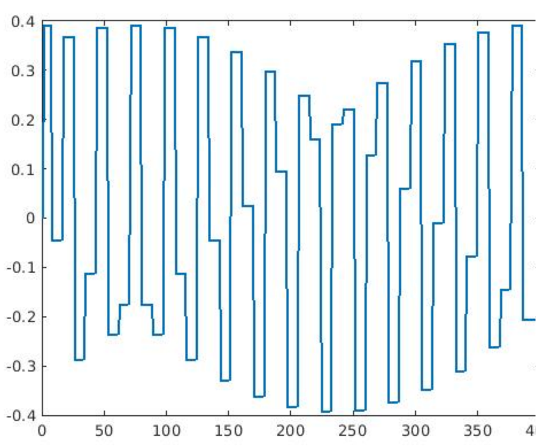
In order to increase the performance, the system has been excited by constantly changing the target attitude. By doing so the control input continuously works on the system, increasing the information in the measured signals. The target attitude and the related input are shown in Figure 4.

The results of the estimation process with this new measured signal are presented in Figure 5. Note that the measured signal vehicles more information than the previous case. For this reason the observer is able to correctly estimate and identify the system.

2.3 Satellite - down sampling

This section discusses another key point in the algorithm tuning, which is the choice of the down sampling windows w and N_{T_s}. Consider the same framework described in subsection 2.2, with the control input, and assume to set:

$$\begin{aligned} w &= 5, \\ N_{T_s} &= 20 \end{aligned} \tag{2.6}$$


(a) Target attitude

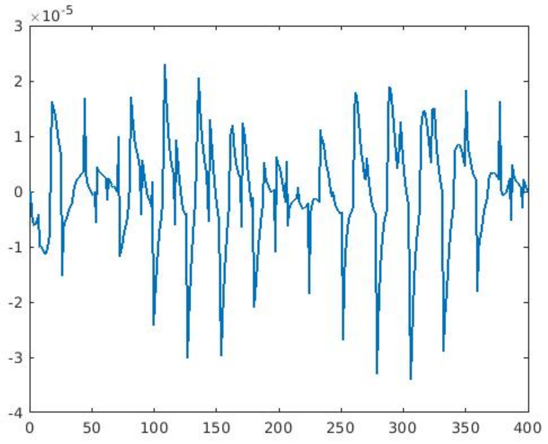
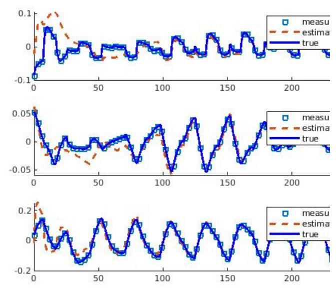
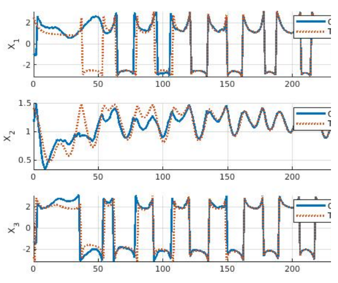

(b) Control input

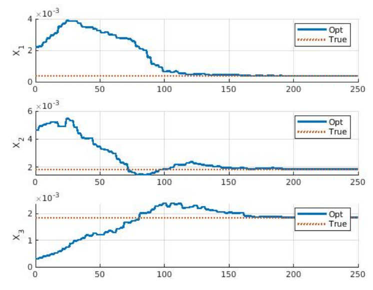
Figure 4: Persistently exciting input



(a) Windowed data



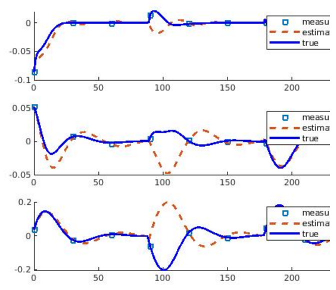
(b) Euler angles



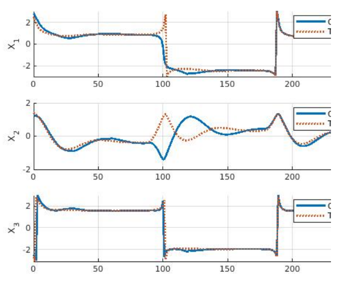
(c) Parameters

Figure 5: Results - with pe input

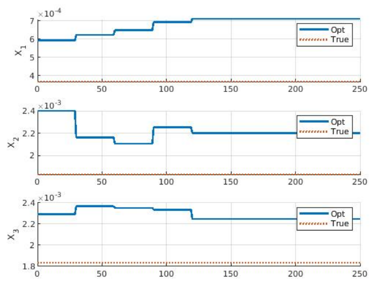
The results are presented in Figure 6. The distance between the measures is too big, resulting in a poor description of the signal. Compare this result with the down sampling used in Figure 5: the signals are well described by the buffer and the inter-samples.



(a) Windowed data



(b) Euler angles



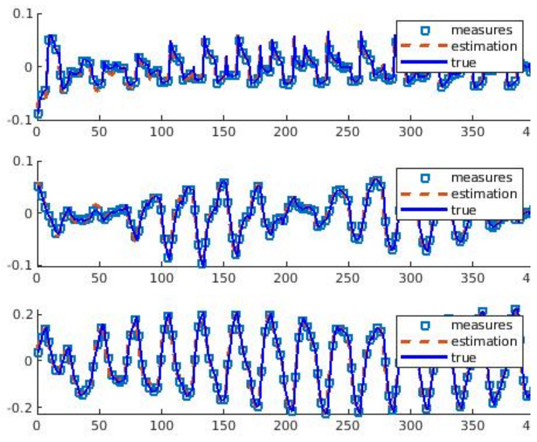
(c) Parameters

Figure 6: Results - down sampling

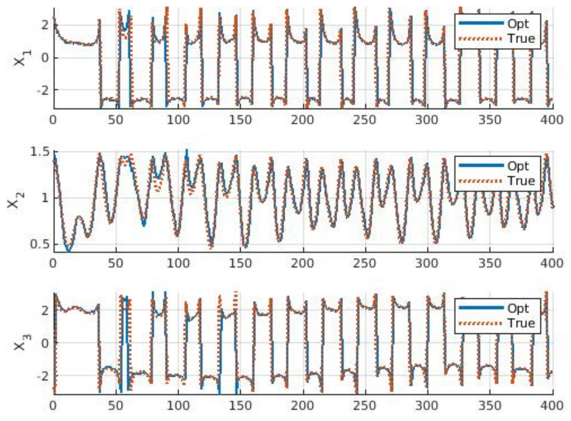
2.4 Satellite - derivative and fault detection

This section presents the action of the derivative in the cost function and uses this algorithm to detect a fault on the satellite. As reported in subsection 1.2, the time derivative of the measurements can introduce more information about the system, improving the accuracy of the estimation. This is clear if a system like the satellite is considered. By differentiating the angular velocity, the inertia comes in.

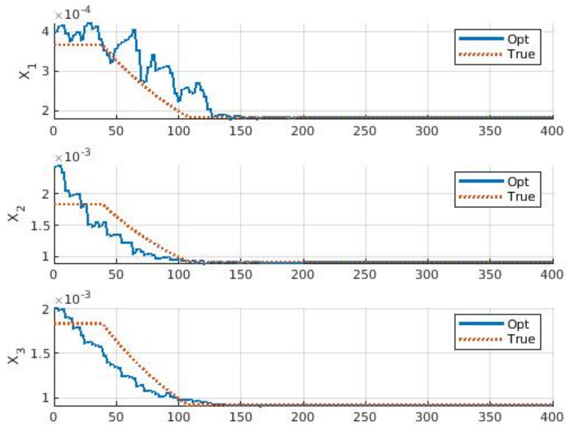
A possible use of this algorithm is on systems fault detection. In the following simulation the satellite undergoes a fault causing a loss of inertia (e.g. component detachment). In Figure 7 the performance of the algorithm is assessed in presence of the derivative in the cost function. Proper down sampling and persistent exciting input are considered. The state estimation works properly, and the parameters are correctly identified.



(a) Windowed data



(b) Euler angles



(c) Parameters

Figure 7: Results - fault detection

2.5 Runaway model

The algorithm has been also tested on the runaway electrons model. Runaway electrons are a phenomenon which develops within plasma currents. They consist in high energy electrons leading to periodic bursts. These bursts need to be controlled in order to avoid damages to the surrounding structure. The model is described by the following equations:

$$\begin{aligned}\dot{x}_1 &= -2x_1x_2 - 2S + Q \\ \dot{x}_2 &= -\nu x_2 + \gamma(x_1x_2 + S) - \gamma_1 \frac{x_2}{1 + \dfrac{x_2}{W_t}}\end{aligned} \tag{2.7}$$

For a detailed description of the variables see [1]. The measured state is assumed to be the following:

$$y = x_2 \tag{2.8}$$

The derivative $\dot{y}$, if used, will be numerically evaluated. The goal of the algorithm is to estimate also the correct values of (γ, γ_1); for this reason the state will be expanded as follows:

$$x = \begin{bmatrix} x_1 & x_2 & \gamma & \gamma_1 \end{bmatrix}^T \tag{2.9}$$

The model considered is updated as follows:

$$\begin{aligned}\dot{x}_1 &= -2x_1x_2 - 2S + Q \\ \dot{x}_2 &= -\nu x_2 + \gamma(x_1x_2 + S) - \gamma_1 \frac{x_2}{1 + \dfrac{x_2}{W_t}} \\ \dot{x}_3 &= 0 \\ \dot{x}_4 &= 0\end{aligned} \tag{2.10}$$

The simulation has the following parameters set:

– Sampling time: $T_s = 5e - 4s$

– Down sampling: $w = 20, N_{T_s} = 10$

– Default optimisation: `fminunc` = gradient based method without constraints.

2.5.1 `fminunc` - no derivative

This section presents the results obtained by running the observer on the runaway model described in Equation 2.11. The target data have been generated through the very same model, while the observer started from a corrupted initial condition. On each measure y an additive noise has been introduced to mimic a more realistic behaviour.

The system initial conditions are the following:

– down sampling:

$$w = 10, \tag{2.11}$$
$$N_{T_s} = 20 \tag{2.12}$$

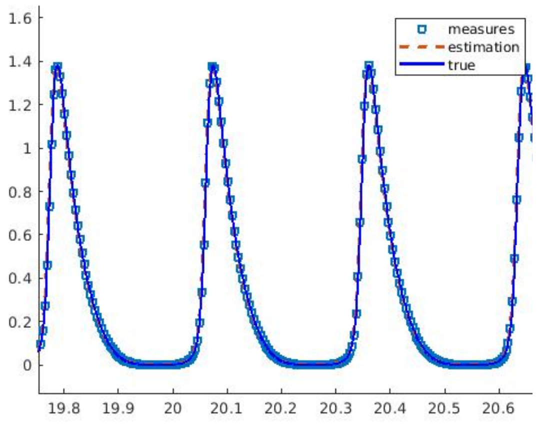
– Correct and corrupted initial condition:

$$x_0 = \begin{bmatrix} 0 \\ 0.001 \\ 0.5 \\ 2.5 \end{bmatrix}, \ \hat{x}_0 = \begin{bmatrix} -0.0001 \\ 0.0006 \\ 0.5004 \\ 2.4886 \end{bmatrix} \tag{2.13}$$

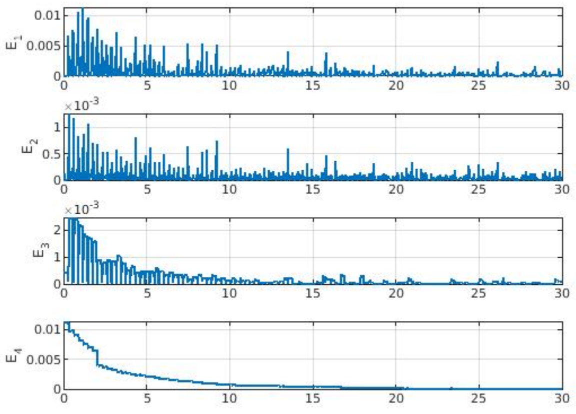
– additive noise:

$$\delta_y = \mathcal{N}(0, 1e-4) \tag{2.14}$$

This simulation doesn't consider the measurement derivative. It takes $30s$ in order to correctly estimate the parameters, as reported in Figure 8.



(a) Windowed data



(b) Optimisation error

Figure 8: Results - `fminunc` without derivative

The optimisation error mean is the following:

$$\bar{e} = \begin{bmatrix} 3.7966e-04 \\ -9.3456e-06 \\ 2.2238e-04 \\ 1.3079e-03 \end{bmatrix} \tag{2.15}$$

2.5.2 `fminunc` - with derivative

This section presents the results obtained by running the observer on the same framework of subsubsection 2.5.1. However, this time the measurement derivative has been exploited during the estimation process.

The system initial conditions are the following:

– down sampling:

$$w = 10, \tag{2.16}$$

$$N_{T_s} = 20 \tag{2.17}$$

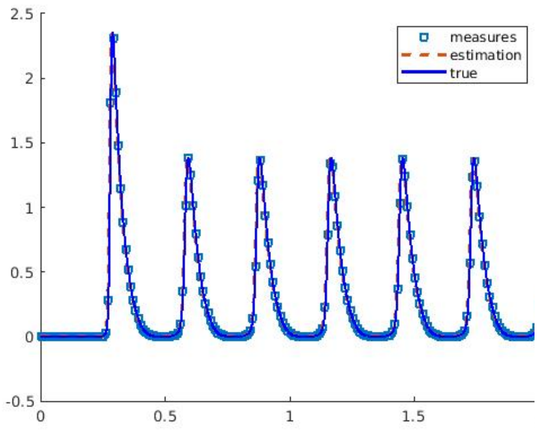
– Correct and corrupted initial condition:

$$x_0 = \begin{bmatrix} 0 \\ 0.001 \\ 0.5 \\ 2.5 \end{bmatrix}, \quad \hat{x}_0 = \begin{bmatrix} 0.0005 \\ 0.0016 \\ 0.5027 \\ 2.4777 \end{bmatrix} \tag{2.18}$$

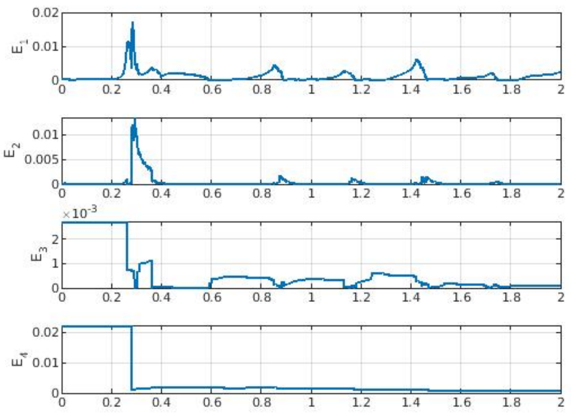
– additive noise:

$$\delta_y = \mathcal{N}(0, 1e-4) \tag{2.19}$$

By using the measurement derivative it takes only $2s$ to correctly estimate the parameters, as reported in Figure 9.



(a) Windowed data



(b) Optimisation error

Figure 9: Results - `fminunc` with derivative

The optimisation error mean is the following:

$$\bar{e} = \begin{bmatrix} 7.5779e-04 \\ 2.2452e-06 \\ 3.3596e-03 \\ 1.9828e-02 \end{bmatrix} \tag{2.20}$$

The estimation error on the state is lower but has some delay, that's the reason for the spikes. In conclusion, the derivative action speeds up the estimation process. However, weighting too much the derivative could lead to instability in the estimation process.

2.5.3 Sensitivity equations

The algorithm has been applied also with the sensitivity equations method, considering only the following state and output:

$$\begin{aligned} x &= \begin{bmatrix} x_1 & x_2 & \gamma & \gamma_1 \end{bmatrix}^T \\ y &= x_2 \end{aligned} \tag{2.21}$$

Therefore, no derivative is exploited in the estimation. As described in subsubsection 1.2.3 the computations unfold as follows:

1. Get the measure: each N_{Ts} sampling times get the measures and store them in the buffer.
2. Run the gradient descent: Equation 1.8 turns out to be described by the following terms:

$$\begin{aligned} y(t_i) &= x_2(t_i) \\ h(\xi_1(t_i)) &= \hat{x}_2(t_i) \\ \frac{\partial h}{\partial x} &= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \\ \frac{\partial f}{\partial x} &= \begin{bmatrix} -2x_2 & -2x_1 & 0 & 0 \\ \gamma x_2 & -\nu + \gamma x_1 - \gamma_1 \dfrac{1}{1+\dfrac{x_2}{W_t}} + \gamma_1 \dfrac{x_2}{W_t\left(1+\dfrac{x_2}{W_t}\right)^2} & x_1 x_2 + S & -\dfrac{x_2}{1+\dfrac{x_2}{W_t}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{2.22}$$

   (ξ_1, ξ_2) are propagated from the initial condition as in Equation 1.10 and used to compute each term of $\dfrac{\partial J}{\partial x_0}$ in the summation of Equation 1.8.
3. Gradient descent: lastly, the optimisation algorithm is implemented as in in Equation 1.11.

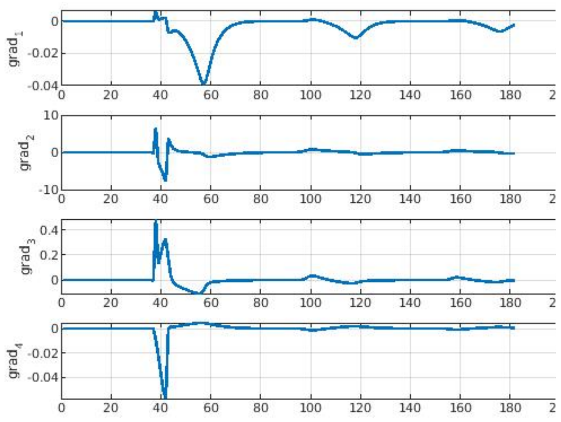
Results are presented in Figure 10. The algorithm works in term of estimation but its identification performances are poor. This is because of the gradient structure. Adding derivatives could improve but this has still to be done.

2.5.4 `fminunc` - experimental data

This section presents the results obtained by running the observer on a batch of data retrieved from a real Tokamak and describing runaway electrons bursts.

In figure Figure 11 the set of data is compared to the synthetic data described by the model in Equation 2.11. In each burst occurrence the experimental data show some lower peaks after the major one. This behaviour is not well described by the current model. As a consequence, we expect the estimation procedure not to work properly on this dataset.

A first attempt has been done on the raw data and with the following setup:


(a) Windowed data

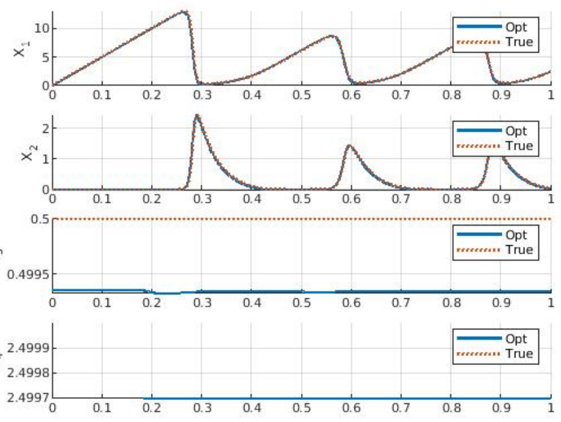
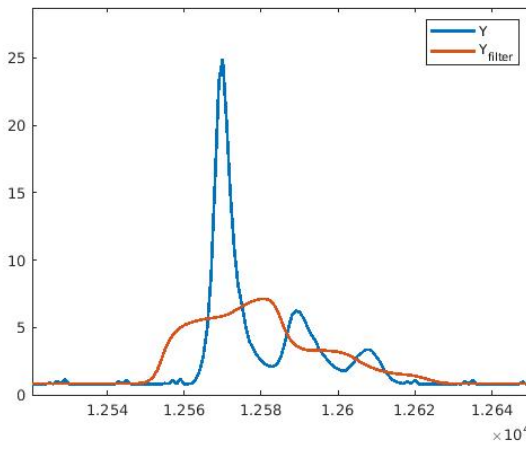

(b) Optimisation error

Figure 10: Results - `fminunc` with derivative


(a) Real data

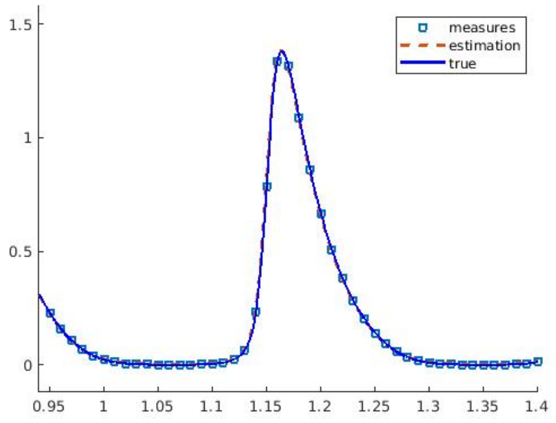

(b) Synthetic data

Figure 11: Real VS synthetic data

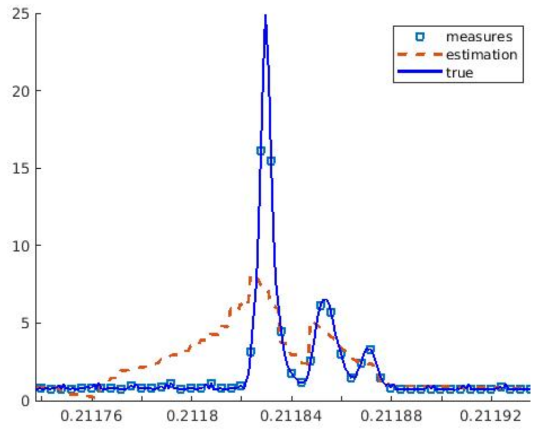
– down sampling:

$$w = 20, \tag{2.23}$$
$$N_{T_s} = 4 \tag{2.24}$$

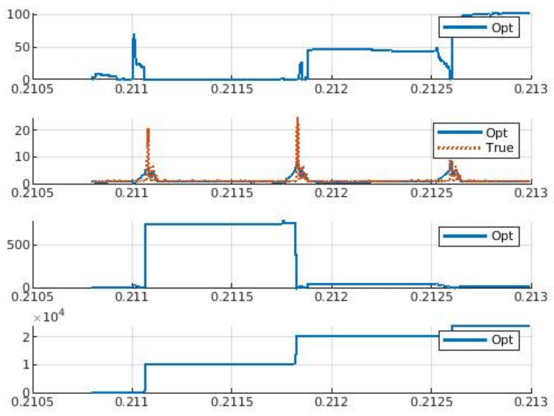
– Sample time: $T_s = 1e - 6s$

As expected the observer can't fit properly neither the state or the parameters. Results in Figure 12

A second attempt has been done by filtering the raw data with a moving average in order to cancel the lower peaks, still with poor results. In conclusion the current model can't fit the real data, but the main issue is probably in the equations rather than in the algorithm procedure.



(a) Real data - window



(b) Real data - estimation

Figure 12: Observer results

References

[1] Paolo Buratti. "runaway model stability analysys". In: () (cit. on p. 10).

[2] Hassan K. Khalil. *Nonlinear Systems, 3rd edition* (cit. on p. 3).