Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: LISA Pathfinder modeling

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Gonzalo Moreno Pousa

DIRECTOR: Marc Díaz-Aguiló, Enrique García-Berro

DATA: 30 d'octubre de 2012

Título: LISA Pathfinder modeling

Autor: Gonzalo Moreno Pousa

Director: Marc Díaz-Aguiló

Fecha: 30 d'octubre de 2012

Resumen

LISA (Laser Interferometer Space Antenna) es un proyecto de la ESA y la NASA que está compuesto por tres satélites que forman un triángulo equilátero orbitando alrededor de la Tierra, diseñado con el objetivo de captar ondas gravitatorias. Para probar su tecnología en el espacio, se ha desarrollado el LPF (LISA Pathfinder), el cual simula un brazo de la constelación del LISA y lleva dos cargas útiles: el LTP (LISA Technology Package) y el DRS (Disturbance Reduction System). A fin de simular el comportamiento de dicha carga útil se ha desarrollado un herramienta de Matlab, llamada LTPDA (LTP Data Analysis toolbox), con el fin de modelar todos los subsistemas del LTP antes del vuelo.

El principal objetivo de este trabajo es medir la precisión con la que la LTPDA modela, y verificar que las hipótesis y los diseños de la herramienta están dentro del margen de precisión necesario, comparado con los métodos tradicionales, los cuales son lentos pero están verificados. En este trabajo se hace primero un análisis de los subsistemas FEEPS (Field Emission Electric Propulsion) e IFO (Interferometer) para comprobar que nuestro modelo es correcto. Hemos realizado simulaciones, tanto en el dominio del tiempo como en el de la frecuencia, usando tanto el módulo LTPDA como los métodos tradicionales, esto es la función de transferencia de Laplace y el State-Space, y hemos comparado los resultados. Después, hemos seguido el mismo procedimiento pero para el sistema completo del LTP.

Los resultados obtenidos de estas simulaciones reflejan, como se esperaba, que el módulo LTPDA es una herramienta verificada que no sólo puede modelar con alta precisión todos los componentes de LPF sino que, además, lo hace con menor esfuerzo computacional y de forma más eficiente.

Title: LISA Pathfinder modeling

Author: Gonzalo Moreno Pousa

Director: Marc Díaz-Aguiló

Date: October, 30th 2012

Overview

LISA (Laser Interferometer Space Antenna) is a joint project of ESA and NASA which consists in three spacecraft forming an equilateral triangle orbiting around Earth, and its goal is to detect gravitational waves. In order to test its feasibility on the outer space a precursor mission has been developed, the so-called LPF (LISA Pathfinder). LPF performs one arm of the LISA constellation and carries two payloads: the LTP (LISA Technology Package) and the DRS (Disturbance Reduction System). To simulate the entire system a Matlab toolbox, the so-called LTPDA (LTP Data Analysis toolbox), has been developed. LTPDA aims to model all the subsystems of the LTP before flight.

The aim of this work is to measure the accuracy with which the LTPDA models the entire system, and to verify that the assumptions and the designs of the toolbox are within a good accuracy margin compared to traditional methods, which are slower but highly verified. Here we first perform an analysis of the FEEPS (Field Emission Electric Propulsion) and IFO (Interferometer) subsystems to check that our modelling method is correct. We have performed simulations both in the time and frequency domains using the LTPDA toolbox and the traditional method, namely the Laplace transfer function and the state space, and we then have compared the results. After that, we followed the same procedure but for the entire LTP system.

The results obtained from those simulations demonstrate that, as expected, the LTPDA is a verified toolbox that not only can model and perform with high accuracy all the systems of the LPF, but also can do this with a significantly smaller computational load and in a more efficient way.

INDEX

Introduction

The European Space Agency (ESA), in collaboration with NASA, has designed a space mission aimed to detect gravitational waves, called LISA (Laser Interferometer Space Antenna). LISA consists of three spacecrafts forming an equilateral triangle orbiting around Earth's orbit and interconnected by laser links. The main technological challenge of the mission is that there are many tests that have to be performed in orbit to prove its feasibility. This led ESA and NASA to develop the LPF (LISA Pathfinder), a technological demonstrator mission for LISA.This mission simulates one arm of LISA, carrying two payloads: the LTP (LISA Technology Package), which is mainly composed by two test masses and the vacuum chambers, and the DRS (Disturbance Reduction System). LPF is needed to perform some tests in the outer space to ensure the viability of LISA. Nevertheless, these two subsystems need to be tested and modelled before flight. Therefore the LTP science team is implementing a data analysis, modelling and simulation toolbox integrated with Matlab and aimed to verify the correct behavior of the entire mission. This toolbox is called the LTPDA toolbox. The LTPDA is based on state space modeling, which allows working easily with multiple-input multiple-output (MIMO) systems. Also, it can keep save the history all the analysis and processes it went through, so can any information about the objects of the toolbox can be retrieved.

The aim of this work is to measure the accuracy with which the LTPDA makes the modeling and verify that the assumptions and the designs of the toolbox are within a good accuracy margin compared to traditional methods, which are slower but highly verified. To do that we build a simplified one-dimensional model of the various state space models of the LISA Technology Package, such as the FEEPS and IFO. After that, a comparison of the simulations between the two methods, both in frequency and in time domain will be performed and we will deliver quantitative measurement of the comparison of both simulation environments. This comparison and validation work is essential to be sure that the assumptions and modelling efforts carried out by the LTP science team within the LTPDA deliver reliable results.

The plan of this work is the following. In the first chapter the LISA and LPF mission will be explained in some detail. Chapter 2 is devoted to briefly describe of the modeling and the state space of LPF, whereas in chapter 3 the LTPDA is fully described. It follows chapter 4, where we describe the FEEPS and IFO modules and the corresponding simulations. I chapter 5 we describe the same kind of simulations of an assembled model of the complete LTP system and we compare them with the LTP system model of the LTPDA. Finally, the work is closed in chapter 6, where we will analyze the results obtained so far and will draw our conclusions.

Chapter 1. LISA and LPF

The Laser Interferometer Space Antenna (LISA) is a joint ESA and NASA mission consisting in three spacecrafts which form an equilateral triangle orbiting around Earth's orbit at a distance of about 50 million kilometres from it. These spacecrafts will have a separation of 5 million kilometres between them. Due to ESA's budget constraints, the mission has suffered some redefinitions and maybe it will change in a near future, but the concept remains unchanged.

The aim of this mission is to detect gravitational waves in a range of frequencies between 10^{-4} and 10^{-1} Hz. This will allow to study and get information never known before about, for example, the background of the universe, the formation of massive black holes and galaxies.

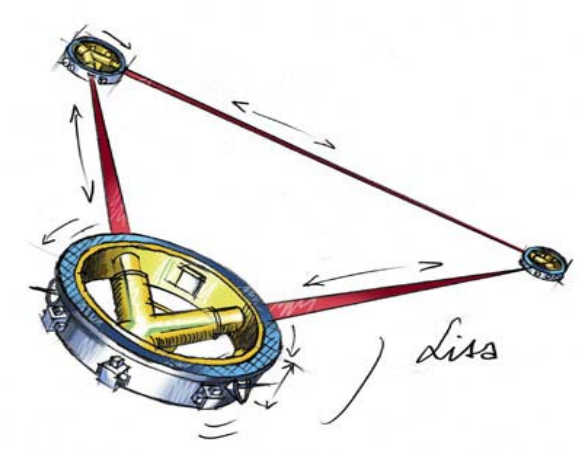

Fig.1.1. Artist impression of LISA

A gravitational wave is a ripple of space-time produced by the fast movement of a massive body. These kinds of waves are transmitted at the speed of light and, until now, it has been impossible to detect directly any of them due to their low amplitude, even though they never stop or slow down. Only violent phenomena in the universe, like the collision of two massive black holes or the remaining radiation of the Big Bang, will produce gravitational waves strong enough to be detected. Therefore, the measurement of these waves will provide a unique vision of the universe in the period before the Dark Age of the universe [1].

LISA is a very challenging mission. Therefore, ESA and NASA decided to undertake a technological demonstrator, called LISA Pathfinder. The LISA Pathfinder (LPF) is one of the European Space Agency’s Small Missions for Advanced Research in Technology (SMART), and it has been joined by NASA. It will test the challenging and critical technologies required for the LISA mission in a flight environment, as they cannot be tested on ground because of the induced noise of the Earth and the lack of a free fall environment. The LPF simulates one arm of the LISA constellation, it will be launched in 2015, and it will start to deliver science results 2-3 months afterwards.

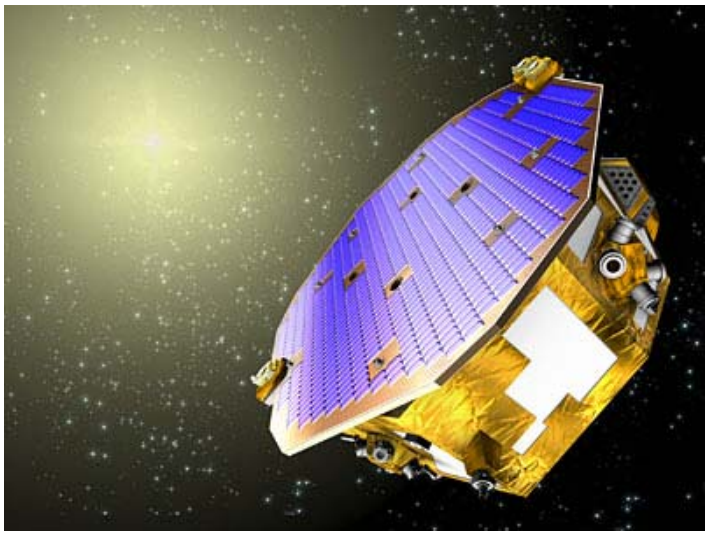

Fig.1.2. Artist impression of LPF

LISA Pathfinder carries two advanced instruments, the LPF and the DRS, which we describe below. The first one is the LISA Technology Package (LTP), which is provided by ESA, and consists in two identical and cubical gold-platinum proof masses of 46 mm suspended separately in two vacuum chambers. They simulate the observational arrangement for the LISA mission, with the difference that the distance between the proof masses is reduced from 5 million kilometres to 35 centimetres. The second one is the Disturbance Reduction System (DRS), which is an experiment provided by NASA's Jet Propulsion Laboratory in California, which includes also a set of micro-rockets that aim to control the spacecraft's position to within a millionth of a millimeter.

Once its technological feasibility would be proven by this mission, the technology on LISA Pathfinder will be ready to be used in the more complex and further-reaching mission LISA. There the relative movement of two spacecrafts located 5 million kilometres apart will be measured to an accuracy of 10 picometers [2,3].

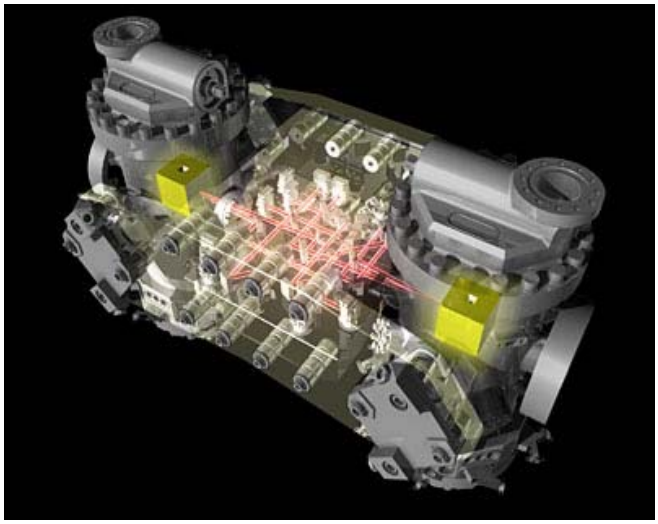

Fig.1.3. Artist impression of the LISA Technology Package

The most important mission goals of LISA Pathfinder are:

1. Demonstrate that the inertial sensor subsystem can put a test mass in a pure gravitational free fall environment only one order of magnitude below the requirements for LISA.

2. Demonstrate that the displacement sensitivity of the laser interferometer with free-falling mirrors is around a factor 5 of the LISA requirements. The strain sensitivity of LISA is around the order of 10^{-21} $Hz^{-1/2}$ at frequencies of ~mHz.

3. Verify if the LISA's critical technologies are ready for a space flight, and if they will last in good conditions all the mission lifetime.

In summary, the final purpose of the mission is to ascertain that the overall physical model of the forces acting on the test mass in the interplanetary space is correct [4]. In this work, we will focus on the LPF and the three major subsystems of the LTP: the inertial sensor subsystem (ISS), the optical metrology subsystem (OMS) and the micro-propulsion subsystem (MPS). These subsystems are briefly described in the following sections.

Inertial sensor subsystem (ISS)

The verification of the correct performance of the ISS is one of the main objectives of the mission because the characteristics of the ISS of LISA Pathfinder will be mainly the same of those of the ISS of the LISA. The inertial sensor subsystem is composed by the two masses, the electrode housing, the vacuum chamber, the caging mechanism, the front end electronics and the UV light unit. The front end electronics is able to actuate and sense the position of the test masses. The UV light unit and the electrode housing are used to control the charge of the test mass due to cosmic ray and solar energetic particle impacts. Even though the inter-planetary space is a good vacuum environment, the spacecraft needs a vacuum chamber owing to the outgassing that fouls its inside and surrounding space. Other issues that must be faced are the launch-vibration conditions and the release requirements for the test mass. These challenges are solved by the three actuators of the caging mechanism assembly. Also, it must be noted that the materials used on the system can not be magnetic, due to its plausible interaction with the interplanetary magnetic field.

Optical metrology subsystem (OMS)

The optical metrology subsystem is a high resolution laser interferometric readout of the test masses' positions. It is composed by the Reference Laser Unit (RLU), the Laser Modulator (LM), the Optical Bench Interferometer (OBI) and the phasemeter. It is in charge of measuring the absolute distance between the test mass 1 and the spacecraft, and the relative one among the two test masses, this last distance is called differential channel.

Micro-propulsion subsystem

The micro-propulsion subsystem harnesses the field emission electric propulsion (FEEP) technology. The MPS is composed by the FEEPS cluster assembly, which consists of 4 FEEP thrusters assembly, the Neutralizer Assembly (NA), which nulls the spacecraft charge imbalance induced by the thrusters, and the Power Control Unit (PCU), which provides power supply and control to the FEEPS cluster, the NA and the telecommand

and telemetry tasks. The goal of this subsystem is to apply a force in order to counteract the external forces acting on the satellite and thus, keep both test masses on a constant free-fall [5].

Chapter 2. Introduction to modelling and State-Space

Older methods and older control techniques were limited to single-input single-output (SISO) systems because they did not use the internal structure knowledge. Thus, if a feedback control is needed, the behaviour of the closed-loop behaviour must be restricted. These methods are usually referred to as a *Transfer Function methods*. Nowadays, new control theories can overcome these restrictions, letting us work with multiple-input multiple-output (MIMO) systems. Among such representations, state space modelling is one of the most powerful techniques to represent MIMO systems of high-order differential equations.

The state-space equations, as they have been called, are a group of coupled first-order differential equations (*state equations*) in a set of internal variables called *state variables*, and algebraic equations that combine the state variables into physical output variables that descript the dynamics of a system. In this work we will focus on systems that are linear and time-invariant (LTI), which are dynamic systems described by linear differential equations with constant coefficients. Such systems can be expressed as:

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t) \tag{2.1}$$

where the state vector $\boldsymbol{x}$ is a column vector of length n, the input vector $\boldsymbol{u}$ is a column vector of length r, **A** is an $n \times n$ square matrix of the constant coefficients a_{ij}, named the “state matrix” and **B** is an $n \times r$ matrix of the coefficients b_{ij} that weight the inputs, named the “input matrix”.

In order to have a complete state-space description, we also need a system output expressed as a set of *n output equations*, where all system variables may be represented by a linear combination of the state variables x_i and the system inputs u_i.

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t) \tag{2.2}$$

where $\boldsymbol{y}$ is a column vector of the output variables, $\boldsymbol{C}$ is an $m \times n$ matrix named the “output matrix” and $\boldsymbol{D}$ an $m \times r$ matrix named the “feed-through matrix”.

In summary, what we need for a complete model of a LTI system are both a set of n state equations (in terms of matrices **A** and **B**), and a set of output equations that relate any output variables of interest to the state variables and inputs (in terms of matrices **C** and **D**). The matrices of the state equations are determined by the system structure and elements. Therefore, they are properties of the system. On the other hand, the value of the output equations matrices depends on the output variables. Then, the modelling procedure may be divided in three simple steps: Determine the order n of the system and the state variables, develop the state equations and the matrices **A** and **B**, and, finally, establish the output equations and matrices **C** and **D**.

The state variables, the value of the variables at initial time t_0, and the system inputs for time $t \geq t_0$ allow us to compute the future system state and outputs for all time $t \geq t_0$, or in other words, the behavior of the system. Also, with these values at any time t, we can obtain the values of all other variables in the system at that time. The minimum number of state variables required to represent a given system, n, is usually equal to the order of the system's defining differential equation. It is important to note that there is no a unique set of state variables that describe a system.

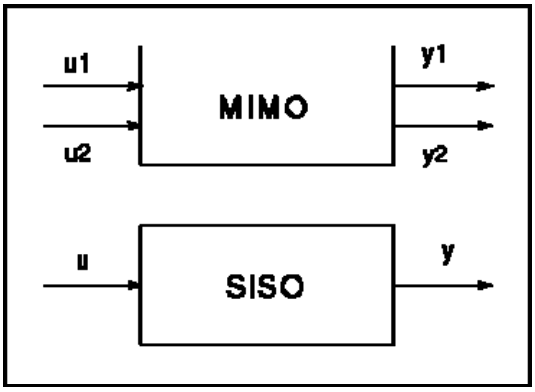

Fig.2.1: SISO and MIMO system

In this work, we will transform from state-space equations to classical form and from classical to state-space form. Therefore, in this section we describe how such transformations will be performed.

From state-space equations to classical form

Using the Laplace transform in a system, its transfer function or its input-output differential equation from the state-space form can be easily obtained. First of all, we need to convert the state equations into a Laplace transform. After that, we have to put all the state variables terms on the left hand side of the equation to solve for those needed for the output variable. Then we substitute these variables in the output equation and identify its transfer function. Finally, we take the transfer function and form the differential equation among the output variable and the system input. For example, let's consider the next state space system:

$$\begin{aligned} \dot{\mathbf{q}}(t) &= \mathbf{A}\mathbf{q}(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}\mathbf{q}(t) + Du(t) \end{aligned} \tag{2.3}$$

Now, take the Laplace Transform (with zero initial conditions since we are finding a transfer function):

$$\begin{aligned} s\mathbf{Q}(s) &= \mathbf{A}\mathbf{Q}(s) + \mathbf{B}U(s) \\ Y(s) &= \mathbf{C}\mathbf{Q}(s) + DU(s) \end{aligned} \tag{2.4}$$

We want to solve for the ratio of Y(s) to U(s), so we need so remove **Q**(s) from the output equation. We start by solving the state equation for **Q**(s):

$$\begin{aligned} s\mathbf{Q}(s) - A\mathbf{Q}(s) &= \mathbf{B}U(s) \\ (s\mathbf{I} - \mathbf{A})\,\mathbf{Q}(s) &= \mathbf{B}U(s) \end{aligned} \tag{2.5}$$

$$\mathbf{Q}(s) = (s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}U(s) = \mathbf{\Phi}(s)\mathbf{B}U(s) \tag{2.6}$$

The matrix $\mathbf{\Phi}(s)$ is called the state transition matrix. Now we put this into the output equation:

$$\begin{aligned} Y(s) &= \mathbf{C\Phi}(s)\mathbf{B}U(s) + DU(s) \\ &= \left(\mathbf{C\Phi}(s)\mathbf{B} + D\right)U(s) \end{aligned} \tag{2.7}$$

So we can solve for the transfer function:

$$H(s) = \frac{Y(s)}{U(s)} = \mathbf{C\Phi}(s)\mathbf{B} + D = \mathbf{C}\left(s\mathbf{I} - \mathbf{A}\right)^{-1}\mathbf{B} + D \tag{2.8}$$

From classical form to state-space equations

The state equations can be used to describe systems that are both continuous and discrete. We will opt to use the generic coefficient matrices *A*, *B*, *C* and *D* for both continuous and discrete systems. The state-space equations can be found directly both from a differential equation and from a transfer function. Some examples will clarify this.

We start showing an example of a differential equation. Let us consider, for example, a third order differential equation,

$$\dddot{y} + a_1\ddot{y} + a_2\dot{y} + a_3 y = b_0 u \tag{2.9}$$

We have the state variables:

$$\begin{aligned} q_1 &= y \\ q_2 &= \dot{y} \\ q_3 &= \ddot{y} \end{aligned} \tag{2.10}$$

And with that, we can assemble the state-space equations for the system:

$$
\begin{aligned}
\dot{q}_1 &= q_2 = \dot{y} \\
\dot{q}_2 &= q_3 = \ddot{y} \\
\dot{q}_3 &= \dddot{y} = -a_3 y - a_2 \dot{y} - a_1 \ddot{y} + b_0 u \\
&= -a_3 q_1 - a_2 q_2 - a_1 q_3 + b_0 u \\
\dot{\mathbf{q}} &= \mathbf{A}\mathbf{q} + \mathbf{B}u = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_3 & -a_2 & -a_1 \end{bmatrix} \mathbf{q} + \begin{bmatrix} 0 \\ 0 \\ b_0 \end{bmatrix} u \\
y &= \mathbf{C}\mathbf{q} + Du = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{q} + 0 \cdot u
\end{aligned}
\tag{2.11}
$$

Now we show an example of a transfer function. Consider this transfer function:

$$
H(s) = \frac{Y(s)}{U(s)} = \frac{b_0}{s^3 + a_3 s^2 + a_2 s + a_1} \tag{2.12}
$$

$$
\left(s^3 + a_3 s^2 + a_2 s + a_1 q_3\right) Y(s) = b_0 U(s) \tag{2.13}
$$

Then, we have the state variables,

$$
\begin{aligned}
Q_1(s) &= Y(s) \\
Q_2(s) &= sY(s) \\
Q_3(s) &= s^2 Y(s)
\end{aligned}
\tag{2.14}
$$

And we only need to take its derivatives and generate the state space system:

$$
\begin{aligned}
sQ_1(s) &= Q_2(s) = sY(s) \\
sQ_2(s) &= Q_3(s) = s^2 Y(s) \\
sQ_3(s) &= s^3 Y(s) = -a_3 Y(s) - a_2 sY(s) - a_1 s^2 Y(s) + b_0 u \\
&= -a_3 Q_1(s) - a_2 sQ_2(s) - a_1 s^2 Q_3(s) + b_0 u \\
s\mathbf{Q}(s) &= \mathbf{A}\mathbf{Q}(s) + \mathbf{B}U(s) == \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_3 & -a_2 & -a_1 \end{bmatrix} \mathbf{Q}(s) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} U(s) \\
Y(s) &= \mathbf{C}\mathbf{Q}(s) + DU(s) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{Q}(s) + 0 \cdot U(s)
\end{aligned}
\tag{2.15}
$$

Further details about these techniques can be found in references [6] and [7].

Chapter 3. LTPDA toolbox and built-in models of the LPF

In this section, we describe the LISA Technology Package Data Analysis toolbox. We present its functionalities and characteristics and also some of its sub-models. The LTP Data Analysis toolbox (LTPDA) is a toolbox within Matlab designed for the analysis of the LISA Technology Package mission data. It is developed by the LTP science team headed by M. Hewitson at the Albert Einstein Institute (AEI) in Hannover [8]. The LTPDA uses data analysis object (AO) which can keep the history of all the processes and operations performed on certain data streams [9]. This toolbox is also able to:

- Generate and process multiple LTPDA objects
- Save and load objects from XML files
- Plot and view the history of the processing in any particular object
- Submit and retrieve objects to/from an LTPDA repository
- Use signal processing capabilities.

An example of analysis object could be:

```
input = ao(pl_w);

>> input
----------- ao 01: sine wave -----------

    name: sine wave
    data: (0,0) (0.1,0.00125663673070233) (0.2,0.00251327147700373) (0.3,0.00376990225450641) (0.4,0.00502652707881886) ...
        -------- tsdata 01 ------------

            fs:  10
             x:  [10000 1], double
             y:  [10000 1], double
            dx:  [0 0], double
            dy:  [0 0], double
        xunits:  [s]
        yunits:  []
         nsecs:  1000
            t0:  1970-01-01 00:00:00.000 UTC
        -------------------------------

    hist: ao / ao / $Id: fromWaveform.m,v 1.36 2011/04/18 16:50:51 ingo Exp $-->$Id: ao.m,v 1.348 2011/05/16 07:15:37 hewitson Exp $
description:
    UUID: 1ddb704d-7dff-4c0c-a912-b88155a88e0d
----------------------------------------
```

In this example a sinusoidal wave with a time of 1000 seconds and a sample frequency of 10 has been considered.

The analysis objects need input parameters to configure their behavior. In the LTPDA these parameters can be set in a parameter list (plist):

```
pl_w = plist('waveform', 'sine wave', 'phi', 0, 'f', 0.002, 'fs', 10, 'nsecs', 1000);

>> pl_w
----------- plist 01 -----------
n params: 5
---- param 1 ----
key:  WAVEFORM
val:  'sine wave'
-----------------
---- param 2 ----
key:  PHI
```

```
val:  0
-----------------
---- param 3 ----
key:  F
val:  0.002
-----------------
---- param 4 ----
key:  FS
val:  10
-----------------
---- param 5 ----
key:  NSECS
val:  1000
-----------------
description:
UUID:
--------------------------------
```

Another important characteristic of the LTPDA toolbox is the built-in representation of certain sub-models that are useful for the analysis of the mission. The LTP science team has developed these models to be capable of simulate/validate the dynamics of the LTP experiment. The LTPDA models must meet some specific requirements [10]:

1. The modelling may be modular
2. The models should be scalable and parametrized
3. The content must be explicit and self-documented
4. The number of manipulations have to be minimized

The kind of modeling that better fulfills all these requirements is the state space modeling, which allows working easily with multiple-input multiple-output (MIMO) systems.

The LTPDA integrates also the possibility of working with parameter estimation techniques. State space models (ssm) will be the object class in the toolbox to represent all the subsystems in the LISA Technology Package. These subsystems are separated in blocks and will be modeled individually to be assembled to produce a model representing the closed-loop form of the LPF in three dimensions. Furthermore, they contain any expected coupling interactions between the different degrees of freedom, so we can use the fully assembled model to see the impact on the x-axis of forces acting on any of the other two axes. In the following section we will describe some of the models of the LTP system [10].

Equations of motion (EOM)

The model for the equations of motion describes the dynamics of the test masses inside the LTP. These dynamical equations are obtained from deriving the states of the position, the attitude, the velocity and the rotation angles of the spacecraft (SC) and the two test masses (TM1 and TM2) in the LTP.

Control chain

The control chain algorithm implements a drag free environment for the test masses. The Drag-Free and Attitude Control System (DFACS), controls the attitude of the spacecraft and also the longitudinal position and angular position of both test masses.

Sensors and Actuators

The sensors model is aimed at simulating the measurements of the instruments of the system’s real displacements and rotations; while the actuators perform the real forces and

the commanded forces affecting the SC and the TMs. The sensors are the Star Tracker, the Electrode Housing and the Interferometer, while the actuators are both the thrusters and capacitive actuators. Except for the FEEPS, which have a low-pass behavior, the rest of the systems are modelled with a simple gain matrix D as they have a much faster dynamics than the 10Hz simulation used in the LTPDA simulator.

Noise sources

As previously mentioned, one of the most important goals of the mission of the LPF is to verify that even in such a noisy environment like the interplanetary medium, it is possible to detect gravitational waves. In order to do that, the science team of the mission needs to characterize and model the different noise sources. In the state space modelling there are three important categories of noise sources: actuation process noise, sensing noise and external process noise. With these models all expected noise sources during the mission can be characterized.

Finally, it must be said that the LTPDA toolbox allows the user simulating a complete LISA Pathfinder system in three dimensions in a fast, easy and accurate way due to the independent analysis of all the subsystems, providing then a modular and flexible tool for data analysis and simulation of any space mission.

Chapter 4. LTP Models Simulation

In this chapter, we present an extensive simulation in one dimension of some LTP models using the LTPDA toolbox and compare the results with the simulations made with the classical Matlab functions. We divide it in two blocks. First of all, we will perform time-domain simulations both for the LTPDA and for the Laplace representations. After this, we do the same but in frequency-domain. Finally, we draw some conclusions about the results of both sections.

Basically, the main goal is to check the validity of the discretization system, therefore we only perform simulations on continuous systems, like FEEPS and IFO. The discrete systems are omitted due to the equivalence, commented on the previous chapter, among the transfer function and the state space method. These systems are discretized with the LTPDA using the function modifyTimeStep, which is basically an integration of their differential equations and a centered Euler difference approximation [11]. They will be compared with the Zero-Order Hold (ZOH), First-Order Hold (FOH) and TUSTIN discretizations of Matlab. The ZOH is a method that holds the input value during the whole sample period, giving a staircase output waveform of the original one. The FOH, on the other hand, takes the derivative of the signal at the time t and guesses where the output signal will be at the time $t+T$ [12]. The TUSTIN method, or bilinear transform, preserves stability and maps every point of the frequency response of the continuous-time filter to a corresponding point in the frequency response of the discrete-time filter. This means that maps the s-plane into the z-plane appropriately and maintaining stability [13].

4.1 Time-domain simulations

First of all we verify that the input for our tests is error-free. To do this, we generate a sinusoidal input, and we compare it with its digitzed version. Then we filter it with each of the systems and we compare the discrete output with the continuous output. This comparison gives us a measure of what is the error due to the digitization process.

In the LTPDA the sinusoidal input is generated using an analysis object (ao) with the next code:

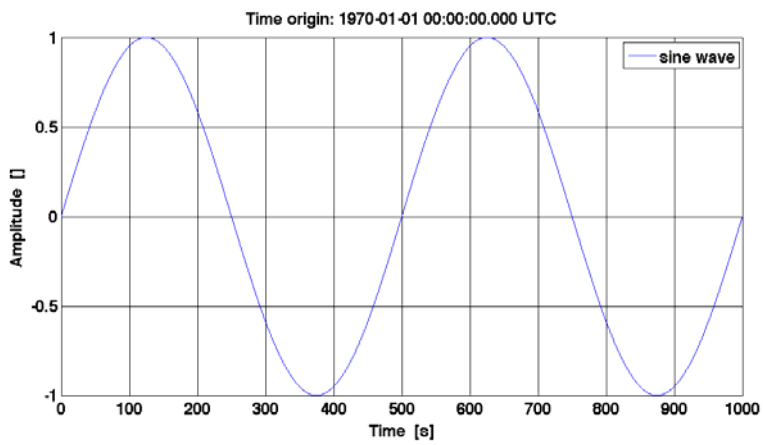

Fig.4.1: Input simulation with LTPDA.

```
pl_w = plist('waveform', 'sine wave', 'phi', 0, 'f', 0.002, 'fs', 10, 'nsecs', 1000);
input = ao(pl_w);
```

The result is shown in Fig. 4 1. Additionally, a normal sinusoidal input is generated using Matlab, with the following piece of code:

```
t = 0:0.1:1000-0.1;
u=sin(2*pi*0.002*t);
```

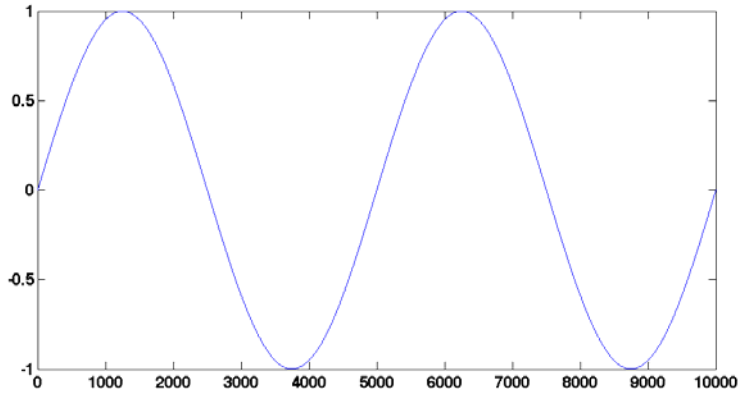

Fig.4.2: Input simulation with Matlab.

The result is shown in Fig. 4.2. A comparison of both inputs shows that there is no error on their performance, as displayed in Fig. 4.3.

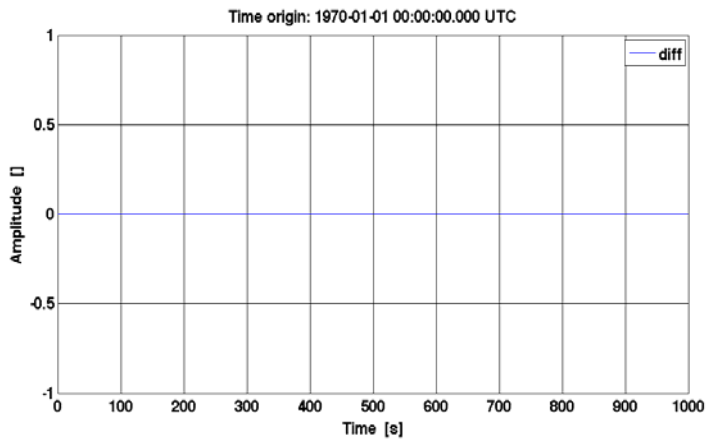

Fig.4.3: Input comparison.

Having performed these preliminary tests and knowing that the error from the input is zero, we are sure that if an error is found, it would be produced by the discretization of the system.

4.1.1 FEEPS

The FEEPS subsystem is composed by two input and two output signals. We generate the state space model using the function:

```
>>feeps = ssm(plist('built-in', 'FEEPS', 'dim', 1))
```

4.1.1.1 LTPDA

Using the *simulate* function and choosing one input and one output signal of the FEEPS, the result is disiplayed in Fig. 4.4 using the function *plot*.[1]

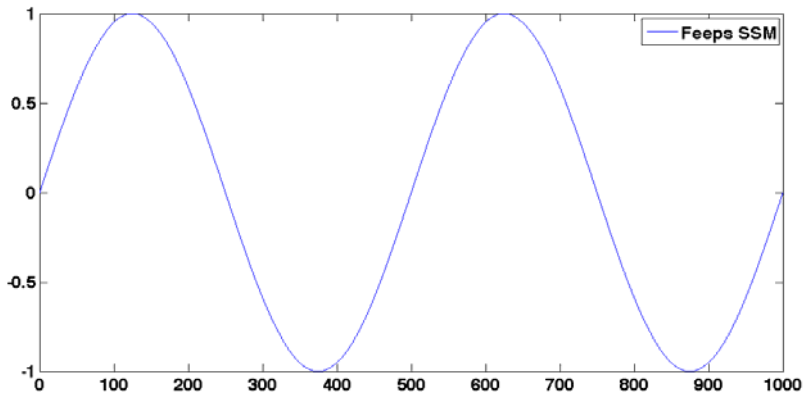

Fig.4.4: Time-domain simulation of the FEEPS system with the LTPDA toolbox, using the DFACS.sc_x, as input, and the FEEPS.x as output.

4.1.1.2 LAPLACE-Transfer function

Here we define the matrices A, B, C and D to generate the state space equation and then convert that into a transfer function, so the functions *ss* and *tf* are used, respectively. We also need to use the same sinusoidal input, and the same system inputs and outputs as the LTPDA form. Unlike in the LTPDA, we can choose the type of discretization used. In our simulations, we use three different types: the ZOH, the FOH and the TUSTIN discretizations, and we use the function *lsim*.

4.1.1.3 Comparisons

The response of the signals it passed through the entire system is displayed in Fig.4.5, where we show from top to bottom the results obtained using the ZOH, FOH and TUSTIN discretizations, respectively. To better see the differences between all thre methods a small code to enlarge the plots (which can be found in the appendix) has been employed. These enlarged plots, showing the regions of interest, are displayed in Figs. 4.6, 4.8 and 4.10 for the ZOH, FOH and TUSTIN discretizations, respectively, while the respective errors are shown in Figs. 4.7, 4.9 and 4.11.

[1]Note: In time-domain simulations it is mandatory to discretize the continuous systems. As the FEEPS is continuous, we have to use the function *modifyTimeStep* before simulate. We use a 0.1 time step.

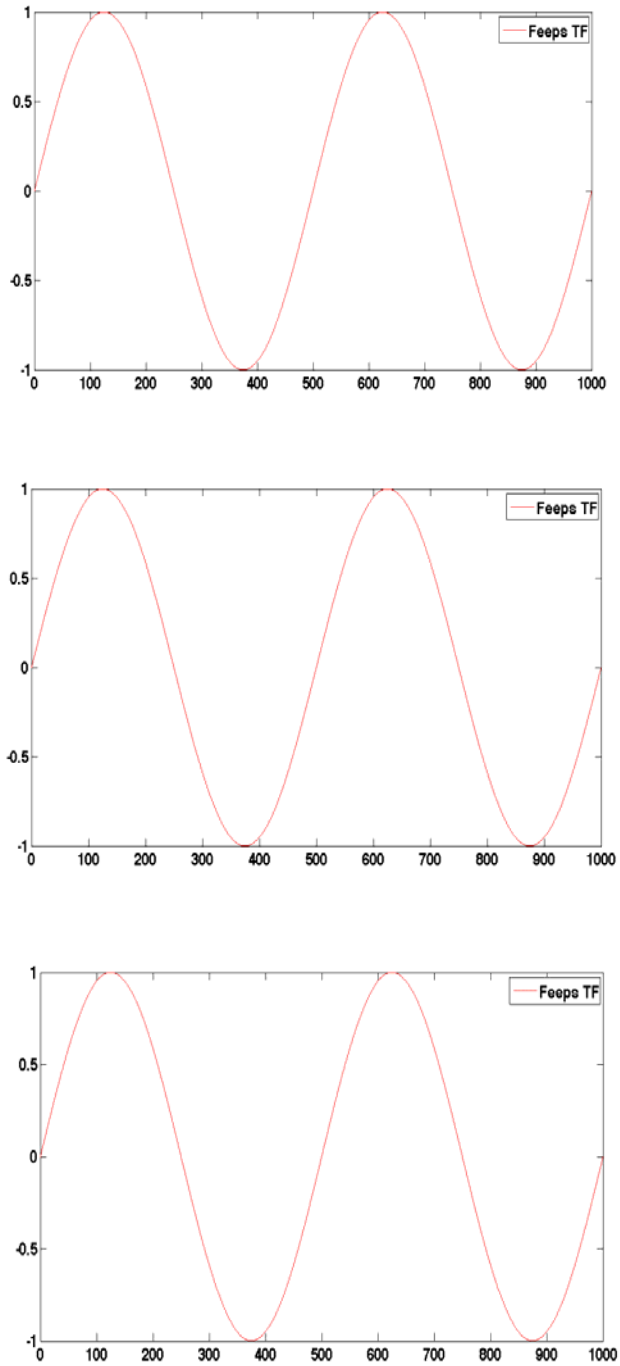

Fig.4.5: ZOH simulation (top), FOH simulation (middle) and TUSTIN (bottom) simulation.

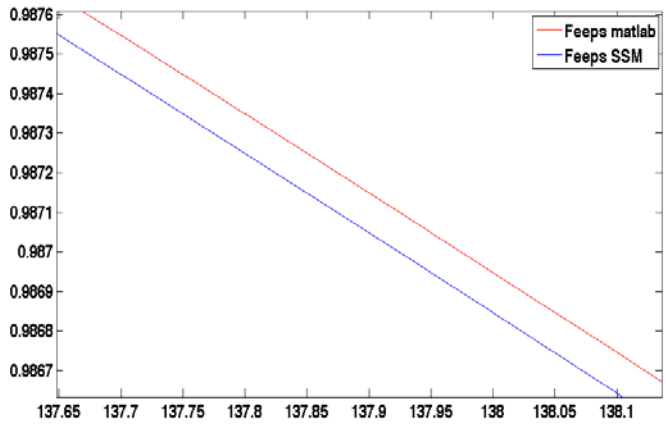

Fig.4.6: LTPDA (blue) versus ZOH discretization (red).

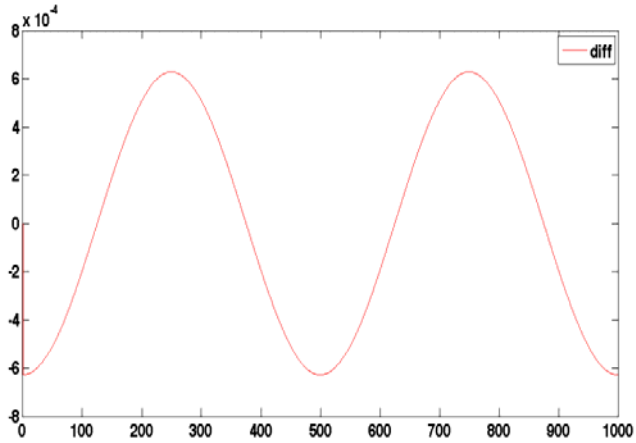

Fig.4.7: Error between the LTPDA and ZOH discretizations.

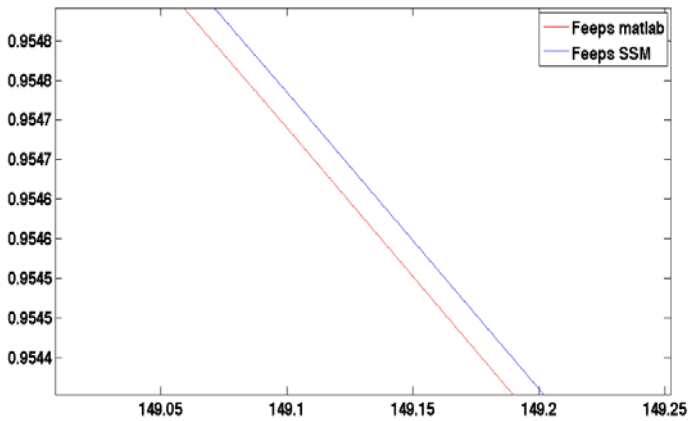

Fig.4.8: LTPDA (blue) versus FOH discretization (red).

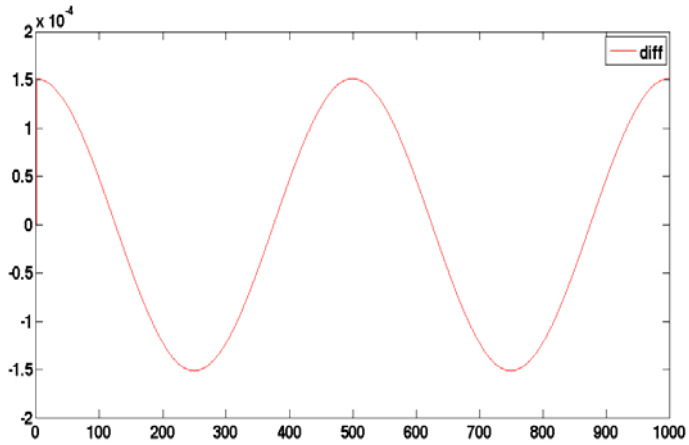

Fig. 4.9: Error between the LTPDA and FOH discretizations.

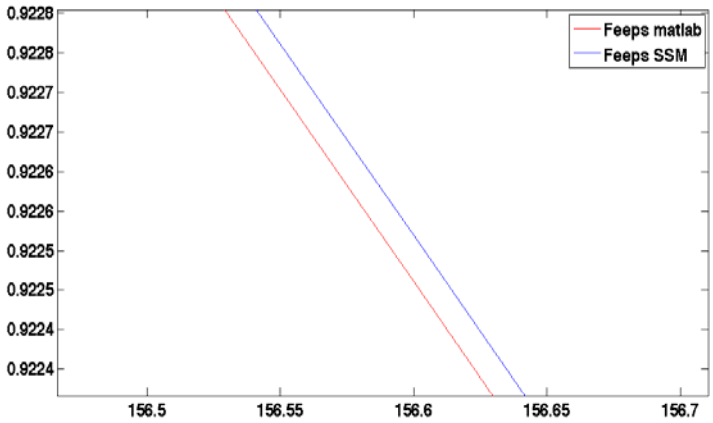

Fig 4.10: LTPDA (blue) versus TUSTIN discretization (red).

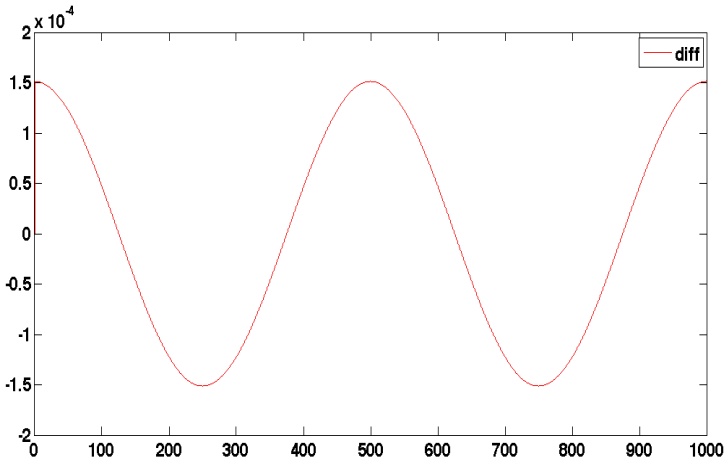

Fig. 4.11: Error between the LTPDA and TUSTIN discretizations.

The mean squared error and the relative error can be also computed and are:

Absolute Error $_{ZOH}$*: 4.44220447e*$^{-6}$ *Relative Error* $_{ZOH}$*: 6.34509586e*$^{-5}$
Absolute Error $_{FOH}$*: 1.06959533e*$^{-6}$ *Relative Error* $_{FOH}$*: 1.52777409e*$^{-5}$
Absolute Error $_{TUSTIN}$*: 1.06949935e*$^{-6}$ *Relative Error* $_{TUSTIN}$*: 2.31966987e*$^{-5}$

As can be seen, the ZOH method has the largest error. This is because it is the simplest (zero order), while the result obtained using the LTPDA agrees very well with both the FOH and TUSTIN discretizations.

4.1.2 IFO

The IFO subsystem is composed by four input and one output signals. We implement the state space model using the function:

```
>> ifo = ssm(plist('built-in', 'IFO', 'dim', 1))
```

4.1.2.1 LTPDA

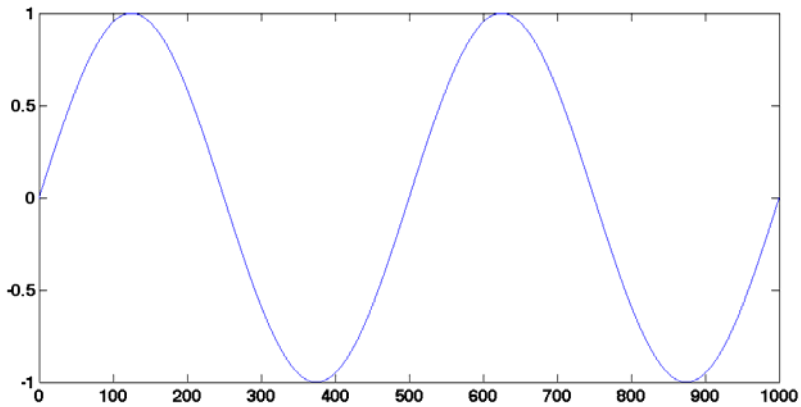

Fig.4.12: Time-domain simulation of the IFO system with the LTPDA toolbox, using the EOM.tm1_x, as input, and the IFO.x1 as output.

4.1.2.2 LAPLACE-Transfer function

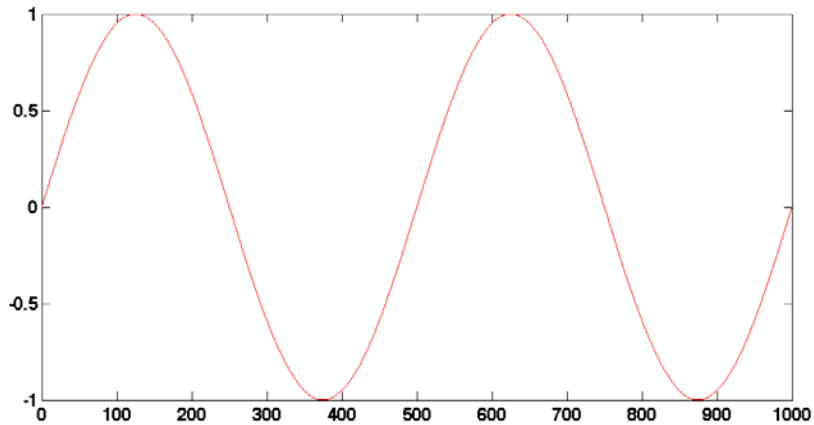

Fig.4.13: ZOH-FOH-TUSTIN simulation.

4.1.2.3 Comparisons

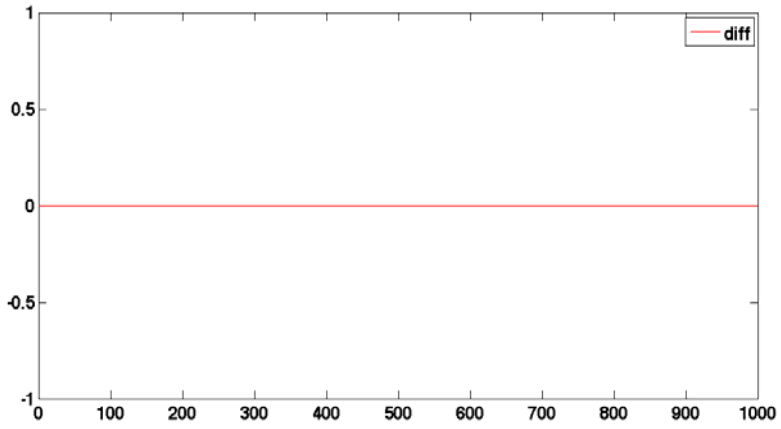

Fig.4.14: Error between the LTPDA and Laplace simulations.

The mean squared error and the relative one are:

Absolute Error $_{ZOH}$*: 0*
Absolute Error $_{FOH}$*: 0*
Absolute Error $_{TUSTIN}$*: 0*

Relative Error $_{ZOH}$*: 0*
Relative Error $_{FOH}$*: 0*
Relative Error $_{TUSTIN}$*: 0*

Here all the results are zero because the A and C matrix are zero. This means, that this is a system without dynamics. Hence, the discretization does not have any effect on the matrices of the system.

4.2 Frequency-Domain simulations

Using the state space model implemented before, we test the frequency response of the same discretized models, FEEPS and IFO, comparing them with the ZOH, FOH and Tustin discretization methods.

4.2.1 FEEPS

4.2.1.1 LTPDA

In the LTPDA, instead of a sinusoidal input as in the time-domain simulations, we need to use a frequency vector:

```
f = logspace(-3, 0, 1000);
```

Then, using the *bode* function and choosing one input and one output signal of the FEEPS, we are able to perform the frequency response:

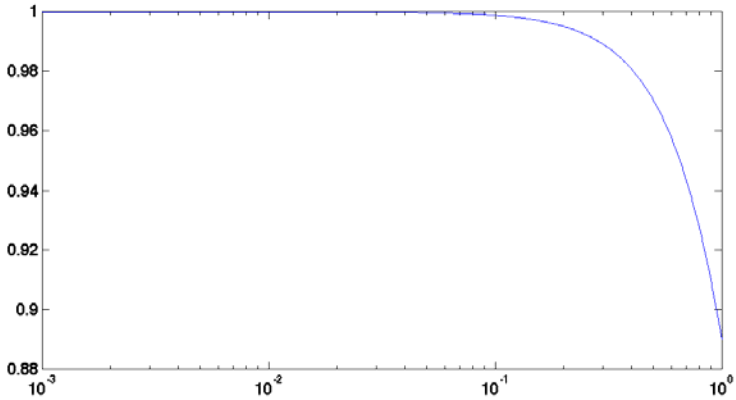

Fig.4.15: Frequency response of the FEEPS system with the LTPDA toolbox, using the DFACS.sc_x, as input, and the FEEPS.x as output.

4.2.1.2 LAPLACE-Transfer function

Here we use the same matrices employed in the time-domain simulations and perform the same procedure with the exception that we will change the *lsim* function to *bode*.

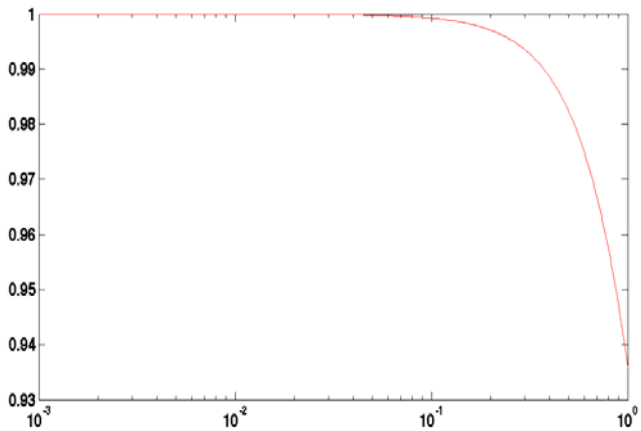

Fig. 4.16: ZOH response.

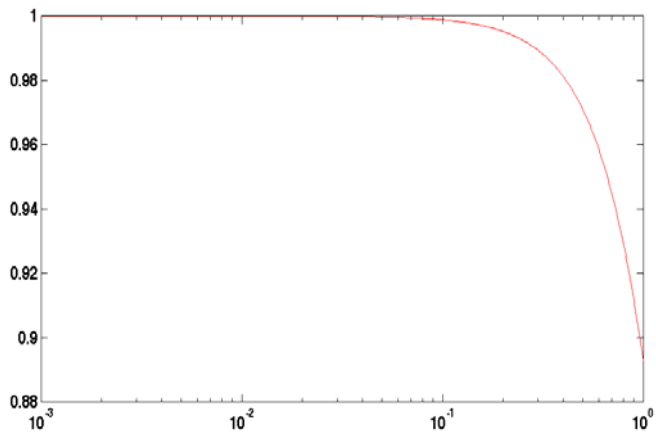

Fig. 4.17: FOH response.

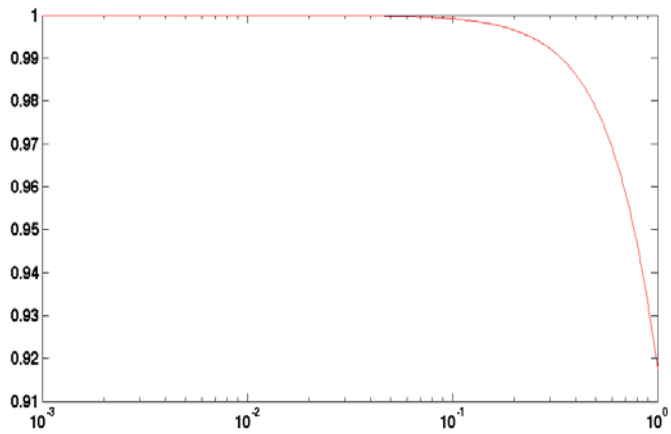

Fig. 4.18: TUSTIN response.

4.2.1.3 Comparisons

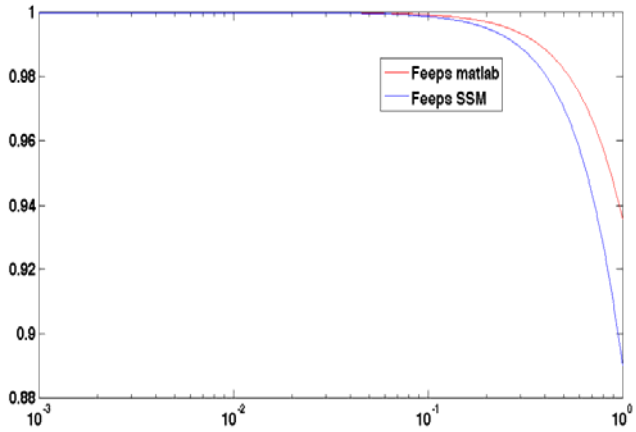

Fig. 4.19: LTPDA (blue) versus ZOH discretization (red).

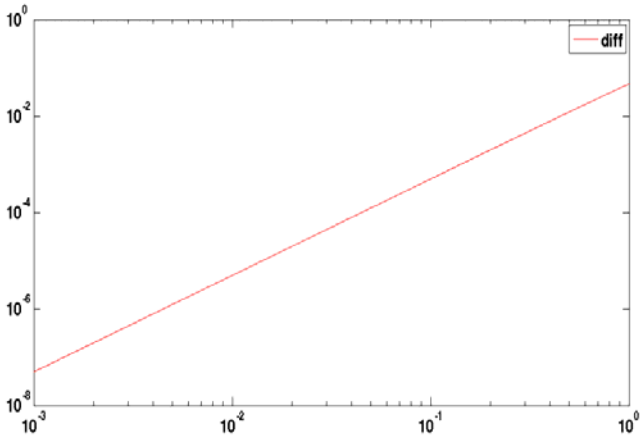

Fig.4.20: Error between the LTPDA and ZOH discretizations.

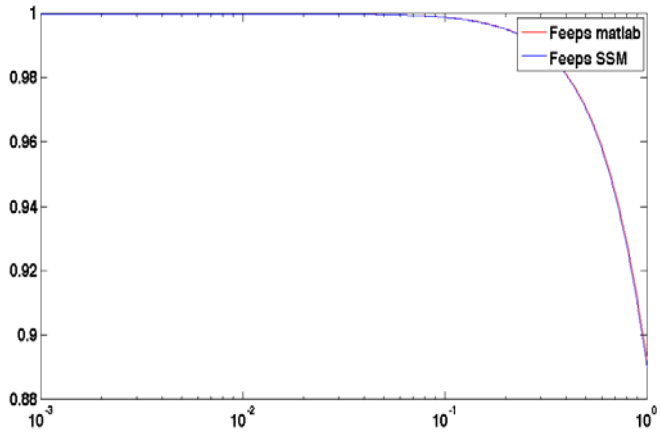

Fig.4.21: LTPDA (blue) versus FOH discretization (red).

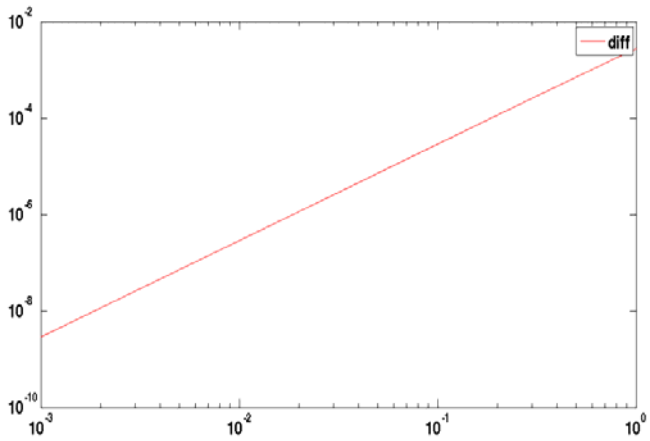

Fig.4.22: Error between the LTPDA and FOH discretizations.

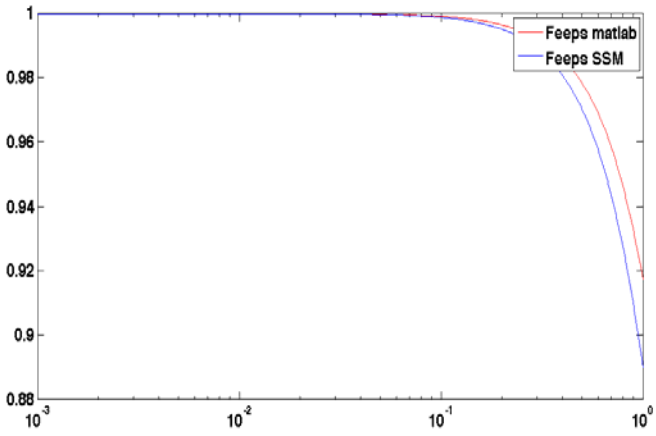

Fig.4.23: LTPDA (blue) versus TUSTIN discretization (red).

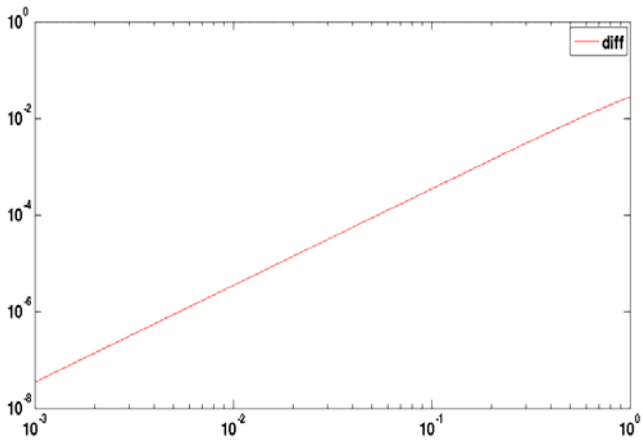

Fig.4.24: Error between the LTPDA and TUSTIN discretizations.

The mean squared error and the relative error are:

Absolute Error $_{ZOH}$*: 0.00028398*
Absolute Error $_{FOH}$*: 1.67257662e*$^{-5}$
Absolute Error $_{TUSTIN}$*: 0.00017885*

Relative Error $_{ZOH}$*: 0.00011025*
Relative Error $_{FOH}$*: 6.46789092e*$^{-6}$
Relative Error $_{TUSTIN}$*: 7.13043863e*$^{-5}$

We find the same behavior previously found in the case of the time-domain simulations. Namely, the ZOH method has the largest error due to its simplicity. However, in this case the FOH method is by far the one which agrees best with the LTPDA frequency response discretization. Additionally, it is worth noting that the differences occur at frequencies higher than 0.1Hz. The LPF experiment is expected to have strong harmonic components within the range of 1e-4 Hz up to 0.01Hz. In this range, the absolute error is below 1e-6. Consequently, the discretization process used by the LTPDA is totally valid.

4.2.2 IFO

4.2.2.1 LTPDA

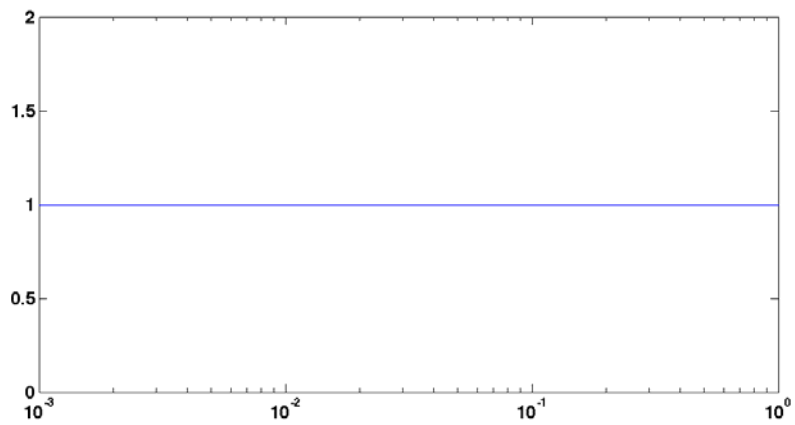

Fig.4.25: Frequency response of the IFO system with the LTPDA toolbox, using the EOM.tm1_x, as input, and the IFO.x1 as output.

4.2.2.2 LAPLACE-Transfer function

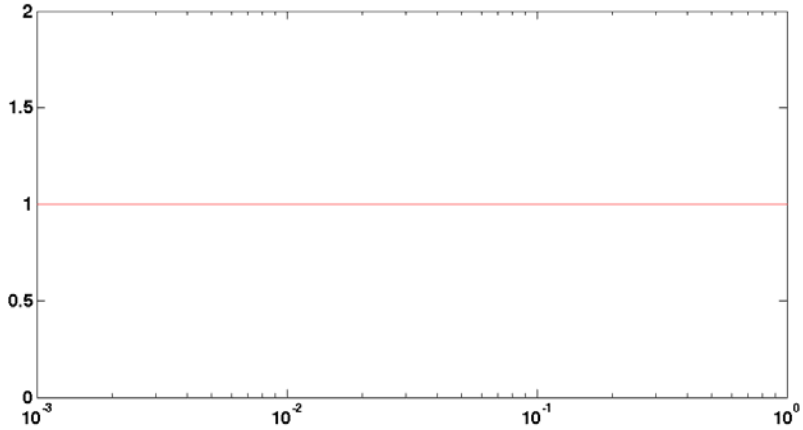

Fig.4.24: ZOH-FOH-TUSTIN frequency response

4.2.2.3 Comparisons

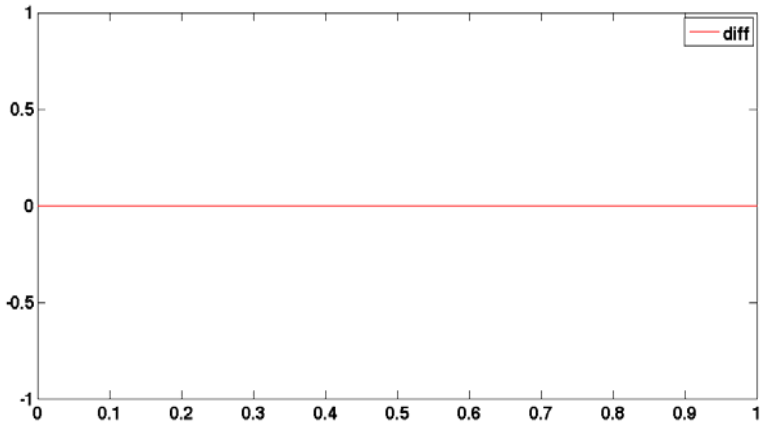

Fig.4.25: Error between LTPDA and Laplace

The mean squared errors and the relative errors are:

Absolute Error $_{ZOH}$*: 0* *Relative Error* $_{ZOH}$*: 0*
Absolute Error $_{FOH}$*: 0* *Relative Error* $_{FOH}$*: 0*
Absolute Error $_{TUSTIN}$*: 0* *Relative Error* $_{TUSTIN}$*: 0*

As commented before, the IFO is a system without dynamics. Thus, the matrices A, B and C are zero. Thus, the discretization has no effect on it.

4.3 Conclusions

To conclude this chapter, it must be said that there is a remarkable difference between the comparisons of the FEEPS and the IFO systems. As can be seen, among the IFO comparisons there is no error on the LTPDA discretization. This

is because the IFO is a simple system with one output, where all the matrices, except D are null, corresponding to a system without dynamics. On the other side, the FEEPS is a complete MIMO system in which all the matrices are non-zero. Hence, even though they are not too complex, they introduce a small error in the simulation. The error for the FOH and TUSTIN methods is quite small in comparison with the error for the ZOH method. This happens because the first two methods are of larger order than the third one. Therefore, the FOH method and the TUSTIN method agree with the desired error margin with the discretization method used in the LTPDA toolbox.

Chapter 5. LTP Simulation

In this chapter, after having simulated the different subsystems of the LTP, we perform the simulations for the entire LISA Technology Package, also in one dimension. The LTP comprises 9 subsystems which can be separated into continuous and discrete systems. The continuous subsystems are the FEEPS, the EOM, the IFO, the ST and the IS; while the discrete ones are the DELAY, the DFACS, the CAPACT and the ACCNOISE. All this results in a very complex system, as the LTP model is composed by twenty one input and sixteen output signals.

In this case, we will compare the LTP from the LTPDA toolbox with a new LTP system generated using the function *assemble*, where all the LTP subsystems are put together. We will develop three different assembled LTP systems, one for each discretization type. In order to do that, we need to generate the matrices for the continuous sub-systems and discretize them with the ZOH, FOH and TUSTIN methods. Having done this, we have to assemble the sub-systems by discretization type and compare the result with the LTP from the LTPDA toolbox.

The assemble instructions used for each of the different discretization systems are:

```
>>ltpZOH = assemble([feepsZOH eomZOH ifoZOH isZOH delay dfacs capact accnoise]);
>>ltpFOH = assemble([feepsFOH eomFOH ifoFOH isFOH delay dfacs capact accnoise]);
>>ltpTUSTIN = assemble([feepstustin eomtustin ifotustin istustin delay dfacs capact accnoise]);
```

We will not use the Star Tracker (ST) in our simulations because in one dimension the ST has neither inputs nor outputs.

5.1 Time-domain simulations

5.1.1 LTP

5.1.1.1 LTPDA

We will use the same sinusoidal input as in the previous chapter:

```
pl_w = plist('waveform', 'sine wave', 'phi', 0, 'f', 0.002, 'fs', 10, 'nsecs', 1000);
input = ao(pl_w);
```

Then, using the *simulate* function and choosing one input and one output signal of the LTP, the result can be ploted with the function *plot*. We have chosen the *DIST_FEEPS.sc_x* as an input, and the *EOM.tm1_x* as an output.

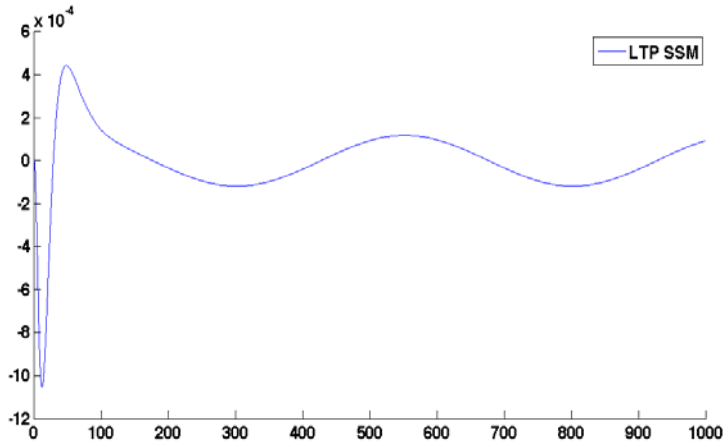

Fig.5.1: Time-domain simulation of the LTP system with the LTPDA toolbox, using the DIST_FEEPS.sc_x, as an input, and the EOM.tm1_x as an output.

5.1.1.2 LTP assembled

As commented before, we will generate the sub-system matrices using different types of discretization and assemble them to perform three different discretized LTP systems. We note that all the matrices and the full m-code can be found in appendix 1. We also use the same sinusoidal input, and the same system inputs and outputs as the LTPDA form. Figure 5.2 shows the ZOH simulation (top panel), FOH simulation (middle panel) and TUSTIN simulation (bottom panel).

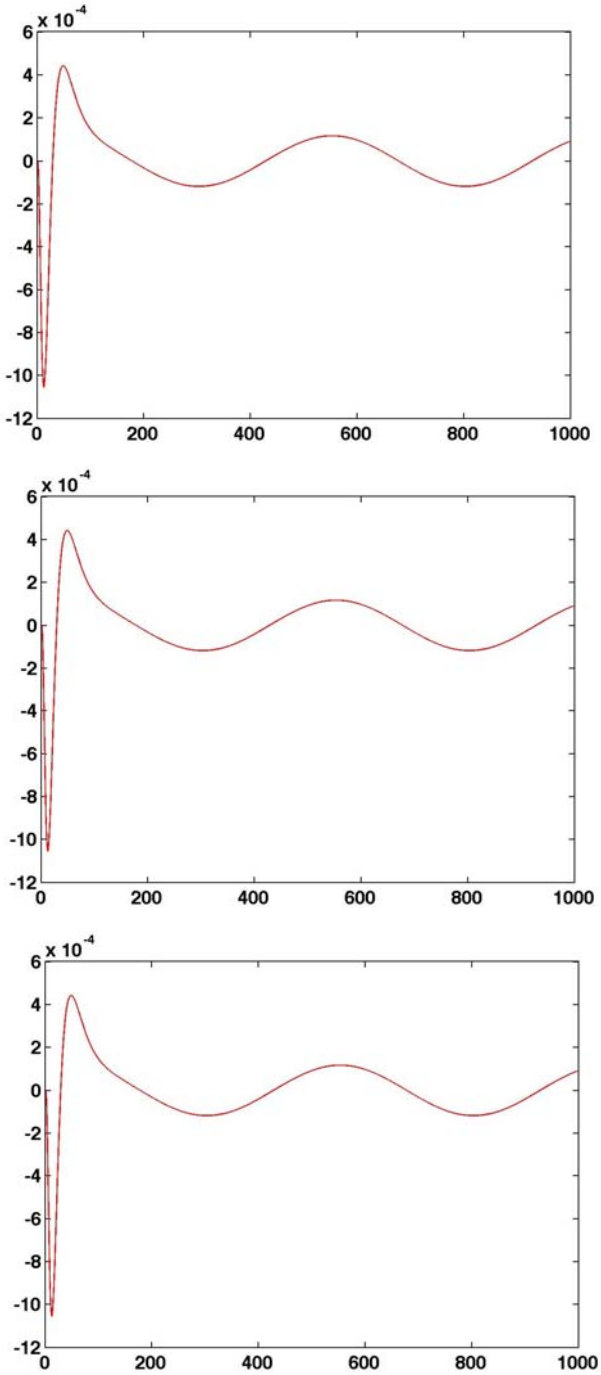

Fig.5.2: ZOH simulation (top-left), FOH simulation (top-right) and TUSTIN simulation.

5.1.1.3 Comparisons

Actually, the differences are very difficult to appreciate. This can be better seen when the error between the different methods is ploted. This is done in figures 5.3, 5.4 and 5.5 for the ZOH, FOH and TUSTIN discretizations, respectively.

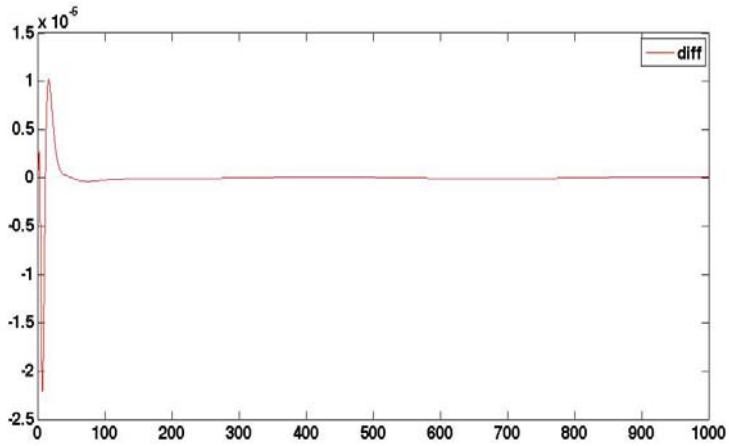

Fig. 5.3: Error between the LTPDA and the ZOH discretizations.

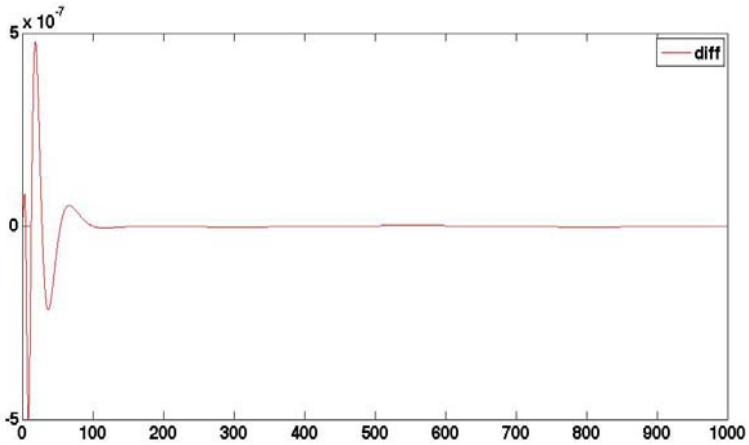

Fig. 5.4: Error between the LTPDA and the FOH discretizations.

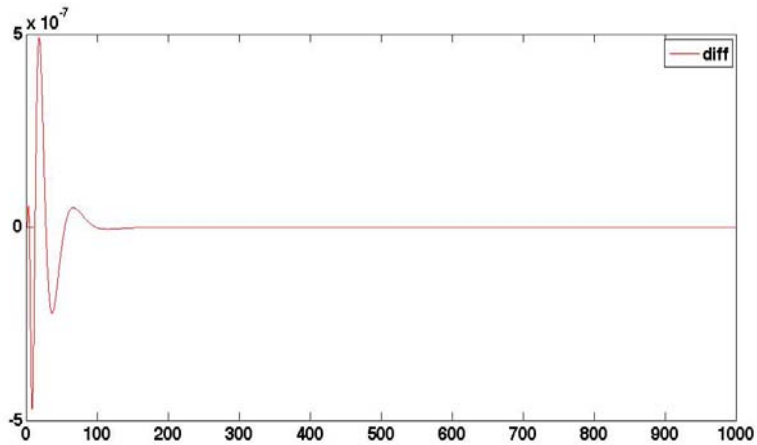

Fig. 5.5: Error between the LTPDA and the TUSTIN discretizations.

The mean squared error and the relative errors are:

Absolute ERROR $_{ZOH}$: 1.62298e^{-08} Relative ERROR $_{ZOH}$: 9.97991e^{-08}
Absolute ERROR $_{FOH}$: 5.65803e^{-10} Relative ERROR $_{FOH}$: 1.09437e^{-09}
Absolute ERROR $_{TUSTIN}$: 5.6629e^{-10} Relative ERROR $_{TUSTIN}$: 1.70062e^{-11}

As can be seen, the ZOH method has the largest error. This is, again, because it is the simplest method (zero order). However, the LTPDA method agrees very well with the FOH and TUSTIN discretizations. Also, the TUSTIN method has the smallest relative error.

5.2 Frequency-domain simulations

Using the state space model generated before, we test the frequency response of the LTP model and compare it with the ZOH, FOH and TUSTIN discretization methods.

5.2.1 LTP

5.2.1.1 LTPDA

We will use the same frequency vector as in the previous chapter:

```
f = logspace(-3, 0, 1000);
```

Then, using the *bode* function and choosing one input and one output signal of the LTP, we the frequency response is shown in Fig. 5.6.

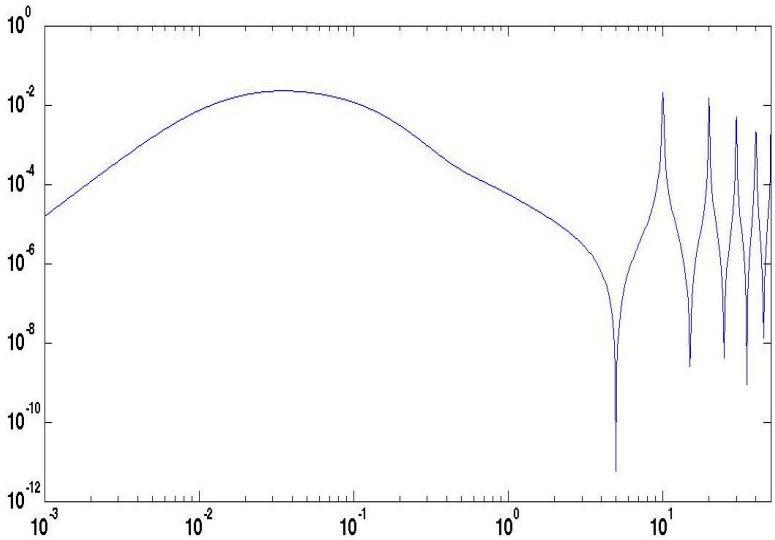

Fig.5.6: Frequency-domain simulation of the LTP system with the LTPDA toolbox, using the DIST_FEEPS.sc_x, as input, and the EOM.tm1_.x as output.

5.2.1.2 LTP assembled

We use the same matrices created in the time-domain simulations and perform the same procedure. The results are shown in Figs. 5.7, 5.8 and 5.9.

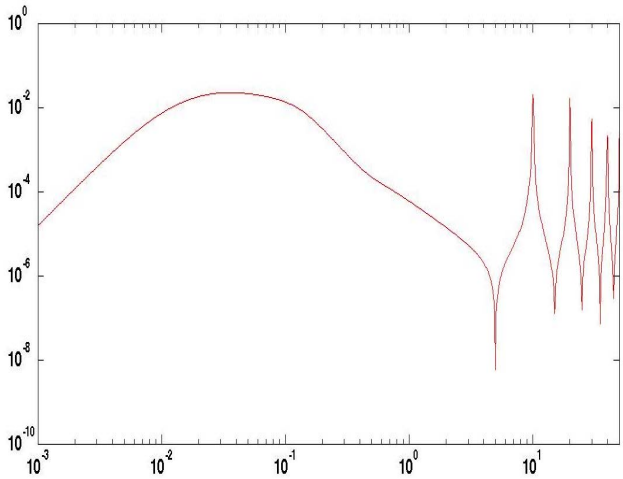

Fig.5.7: ZOH simulation.

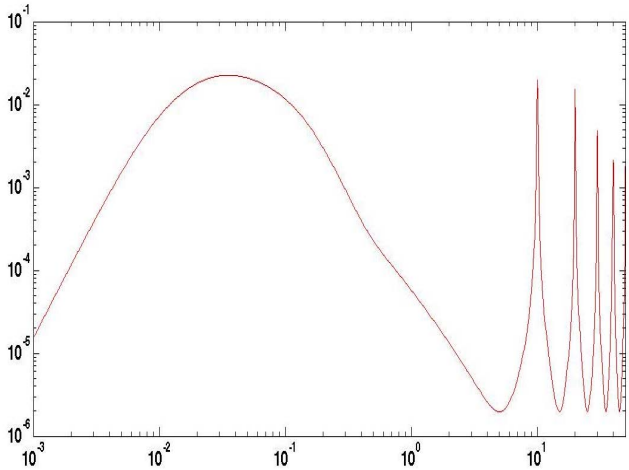

Fig.5.8: FOH simulation.

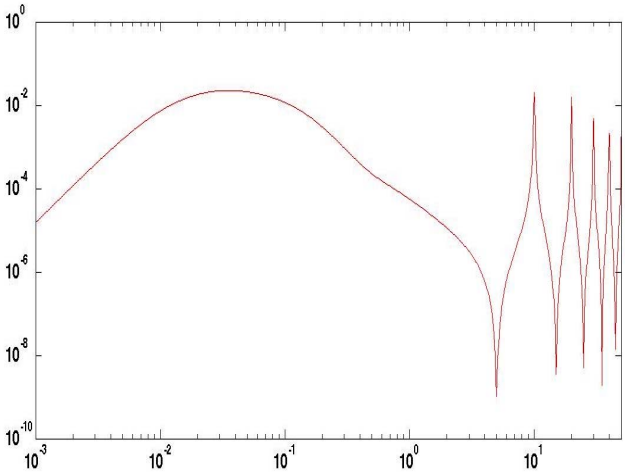

Fig.5.8: TUSTIN simulation.

5.2.1.3 Comparisons

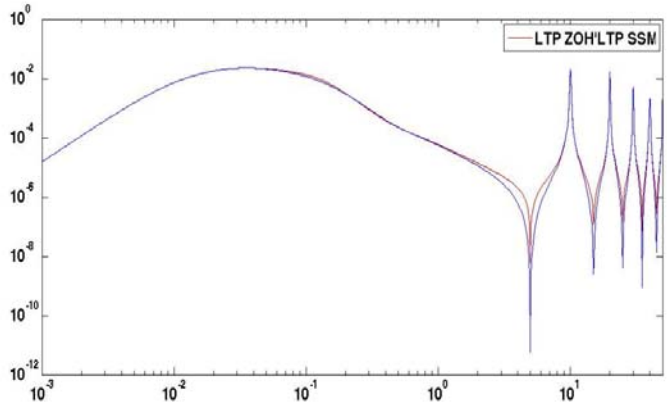

Fig.5.9: LTPDA (blue) versus LTPDA assembled ZOH (red).

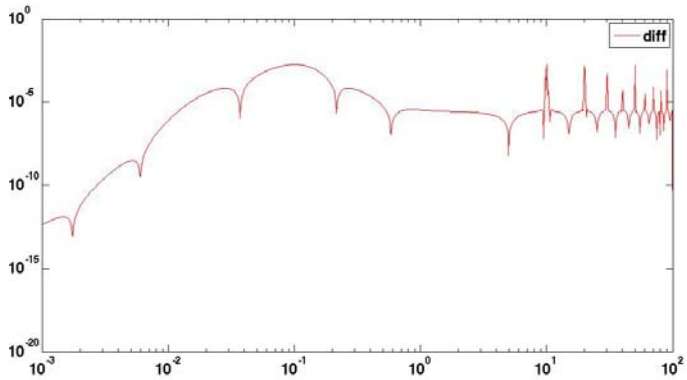

Fig.5.10: Error between the LTPDA and the ZOH discretizations.

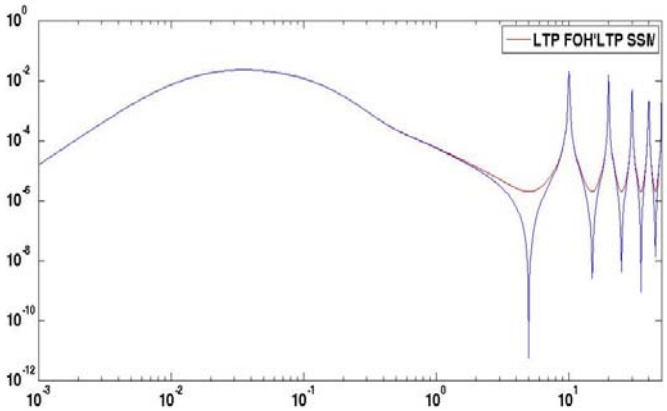

Fig.5.11: LTPDA (blue) versus LTPDA assembled FOH (red).

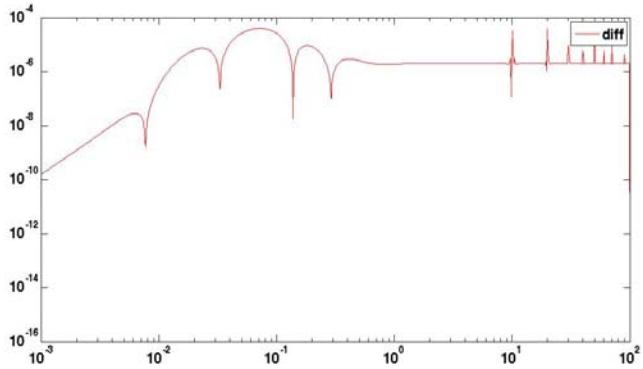

Fig.5.12: Error between the LTPDA and the FOH discretizations.

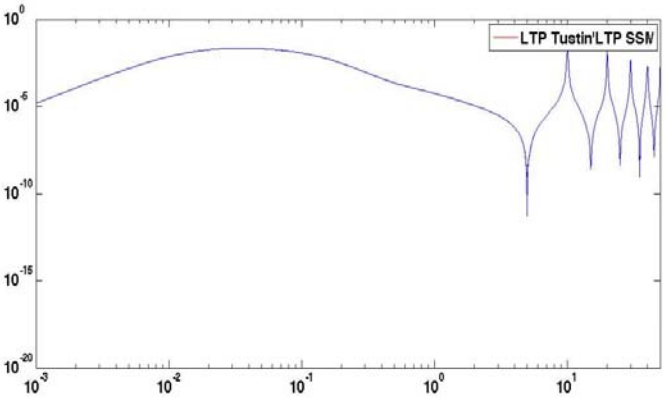

Fig.5.13: LTPDA (blue) versus LTPDA assembled TUSTIN (red).

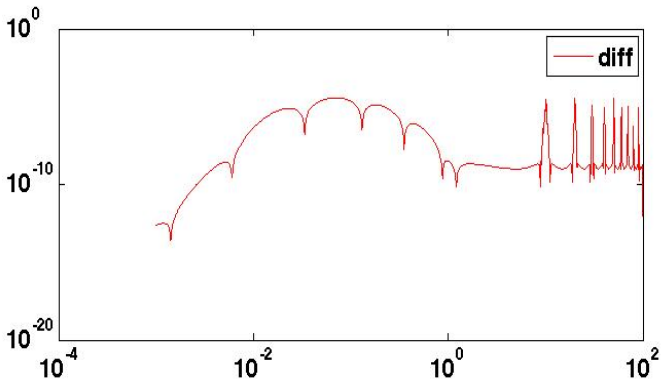

Fig.5.14: Error between the LTPDA and the TUSTIN discretizations.

The thorough comparison of the different discretizations is shown in Figs. 5.9 to 5.14, for the ZOH, FOH, and TUSTIN discretizations, while the mean squared error and the relative error are:

Absolute ERROR $_{ZOH}$: 1.36530e^{-06} Relative ERROR $_{ZOH}$: 3.12933e^{-06}
Absolute ERROR $_{FOH}$: 3.13690e^{-08} Relative ERROR $_{FOH}$: 7.30887e^{-08}
Absolute ERROR $_{TUSTIN}$: 2.92955e^{-08} Relative ERROR $_{TUSTIN}$: 3.55246e^{-08}

Clearly, the ZOH method has the largest error (this, again, is due to its simplicity), whereas the TUSTIN method has the smallest error.

5.3 Conclusions

To conclude this chapter, we mention that the error for the FOH and TUSTIN methods is quite small in comparison with the error obtained when the ZOH method is employed. This is due to the low order of the last method in comparison with the other two. Therefore, the FOH method and the TUSTIN method are perfectly compliant with the output delivered by the discretization process of the LTPDA function.

CHAPTER 6. CONCLUSIONS

The aim of this work was to measure the accuracy with which the LTPDA models the system under study, and to verify that the assumptions and the designs of the toolbox are within a good accuracy margin compared to traditional methods, which are slower but highly verified.

After performing simulations on time-domain and checking the frequency response of subsystems like FEEPS and IFO, and assembling these together with the other necessary subsystems in order to build the complet LTP model, the following tables summarize our results:

	ZOH	**FOH**	**TUSTIN**
FEEPS	*4.44220447e^{-6}*	*1.06959533e^{-6}*	*1.06949935e^{-6}*
IFO	0	0	0
LTP	1.62298e^{-08}	5.65803e^{-10}	5.6629e^{-10}

Table 1. Mean squared error in time-domain.

	ZOH	**FOH**	**TUSTIN**
FEEPS	*6.34509586e^{-5}*	*1.52777409e^{-5}*	*2.31966987e^{-5}*
IFO	0	0	0
LTP	9.97991e^{-08}	1.09437e^{-09}	1.70062e^{-11}

Table 2. Relative error in time-domain.

	ZOH	**FOH**	**TUSTIN**
FEEPS	*0.00028398*	*1.67257662e^{-5}*	*0.00017885*
IFO	0	0	0
LTP	1.36530e^{-06}	3.13690e^{-08}	2.92955e^{-08}

Table 3. Mean squared error in frequency-domain.

	ZOH	**FOH**	**TUSTIN**
FEEPS	*0.00011025*	*6.46789092e^{-6}*	*7.13043863e^{-5}*
IFO	0	0	0
LTP	3.12933e^{-06}	7.30887e^{-08}	3.55246e^{-08}

Table 4. Relative error in frequency-domain.

Additionally, our main conclusions are the following ones:

1) The IFO comparisons have no error because this is a simple system with one output, where all the matrices, except the D one, are null. Thus, the

IFO is a system without dynamics and the discretization does not have any effect.

2) In the case of the FEEPS, it is a complete MIMO system where all the matrices are filled. Hence, even though they are not too complex, when we perform the simulations a small error appears.

3) The error for the FOH and TUSTIN methods is quite small in comparison with the error introduced by the ZOH method. This happens because the first two methods have a larger order than the third one (which is of order zero). This is the reason why the FOH method and the TUSTIN method agree very well with the results obtained using the LTPDA.

To conclude, it must be said that the LTPDA toolbox allows the user simulating a complete LISA Pathfinder system in three dimensions in a fast, easy and accurate way due to the independent analysis of all the subsystems, providing then a modular and flexible tool for data analysis and simulation of any space mission. The main objective, which was the comparison among the LTPDA discretization and the classical methods discretization, has been tested successfully.

The extension of this project has been limited. However some extensions of this work could be the following ones:

1. Perform an extensive simulation on 2 and 3 dimensions.

2. Perform simulations with other inputs.

3. Perform the assembled simulation for the entire LISA Pathfinder (LPF) system and its sub-systems.

References

[1] http://www.esa.int/esaSC/SEMLY2T1VED_index_0.html

[2] http://www.esa.int/esaSC/120397_index_0_m.html

[3] http://www.lisa-science.org/

[4] Vitale, S. et al,. *Science Requirements and Top-Level Architecture Definition for the LISA Technology Package (LTP) on board LISA Pathfinder (SMART-2)* ESA Ref: LTPA-UTN-ScRD Iss3-Rev1

[5] Antonucci, F.et al., *LISA Pathfinder: mission and status*. Class. Quantum Grav. 28 094001 (2011)

[6] http://en.wikipedia.org/wiki/State_space_representation

[7] Rowell, D. 2.14 *Analysis and Design of Feedback Control Systems. State-Space Representation of LTI Systems* (2002)

[8] http://www.lisa.aei-hannover.de/ltpda/

[9] Grynagier, A. and Weyrich, M. *The SSM class: modeling and analyses of the LISA Pathfinder technology experiment*. Stuttgart, August, 2008.

[10] Diaz-Aguiló, M., Grynagier, A., Rais, B., Hewitson, M., Hueller, M., Nofrarias, M., Ferraioli, L., Monsky, A., Congedo, G., Armano, M., Vitale, S., Plagnol, E., Lobo, A and García-Berro, E. *Modeling LISA Pathfinder for Data Analysis.*

[11] Diaz-Aguiló, M., Grynagier, A., Hewitson, M. *A linear MIMO model of LPF implemented in LTPDA*, S2-AEI-TN-3069 (2011)

[12] http://en.wikibooks.org/wiki/Control_Systems/Sampled_Data_Systems#Reconstruction

[13] http://www.mathworks.es/es/help/signal/ref/bilinear.html

List of figures

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

APPENDIX

TÍTOL DEL TFC: LISA Pathfinder modeling

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Gonzalo Moreno Pousa

DIRECTOR: Marc Díaz-Aguiló, Enrique García-Berro

DATA: 30 d'octubre de 2012

Appendix I

In this annex we will show the m-code of the FEEPS and IFO sub-systems, and, also, the code of the complete LTP model, both in time and frequency domain.

FEEPS time-domain m-code

```
%% SSM

ltpda_startup;
%%
clear all

pl_w = plist('waveform', 'sine wave', 'phi', 0, 'f', 0.002, 'fs', 10, 'nsecs', 1000);
input = ao(pl_w);

feeps = ssm(plist('built-in', 'FEEPS', 'dim', 1));

%% CONTINUOUS MATRICES

A=feeps.amats
celldisp(A)

B=feeps.bmats
celldisp(B)

C=feeps.cmats
celldisp(C)

D=feeps.dmats
celldisp(D)

A = [A{1,1}.*ones(feeps.statesizes(1), feeps.statesizes(1)  );]

B = [B{1,1}.*ones(feeps.statesizes(1), feeps.inputsizes(1)  ) zeros(feeps.statesizes(1),
feeps.inputsizes(2)  );]

C = [C{1,1}.*ones(feeps.outputsizes(1), feeps.statesizes(1)  );
     zeros(feeps.outputsizes(2), feeps.statesizes(1)  );]

D = [D{1,1}.*ones(feeps.outputsizes(1), feeps.inputsizes(1)  )
D{1,2}.*ones(feeps.outputsizes(1), feeps.inputsizes(2)  );
     zeros(feeps.outputsizes(2), feeps.inputsizes(1)  )
D{2,2}.*ones(feeps.outputsizes(2), feeps.inputsizes(2)  );]

 %% LAPLACE

feepsSS = ss(A,B,C,D)
[feepsSS,G] = c2d(feepsSS,0.1,'zoh')
tffeeps = tf(feepsSS)

% per agafar entre input 1 i output 1

H = tffeeps(1,1)
t = 0:0.1:1000-0.1;
u=sin(2*pi*0.002*t);

[y,t,x] = lsim(H,u,t)


%% SS

modifyTimeStep(feeps,0.1);
feepsoutSSM = feeps.simulate(plist('AOS VARIABLE NAMES', {'DFACS.sc_x'}, 'AOS',
input,'RETURN OUTPUTS', {'FEEPS.x'},'nsamples', 10000, 'fs', 10));
feepsoutSSM1 = feepsoutSSM.getObjectAtIndex(1);



%% figure


figure
plot(t, y, 'r')
hold on
plot(feepsoutSSM1.x, feepsoutSSM1.y, 'b')
legend('Feeps TF')


figure
plot(t, (y - feepsoutSSM1.y) , 'r')
legend('diff')

%% ERROR

nsamples = 10000;

Abresult = 1/nsamples * sqrt( sum ( (y - feepsoutSSM1.y).^2  )  )
```

```
Relresult = 1/nsamples * sqrt( sum (y - feepsoutSSM1.y).^2/sum(feepsoutSSM1.y).^2)
```

IFO time-domain m-code

```
%% SSM

ltpda_startup;
%%
clear all

pl_w = plist('waveform', 'sine wave', 'phi', 0, 'f', 0.002, 'fs', 10, 'nsecs', 1000);
input = ao(pl_w);


ifo = ssm(plist('built-in', 'IFO', 'dim', 1));

%% MATRICES

A=ifo.amats
celldisp(A)

B=ifo.bmats
celldisp(B)

C=ifo.cmats
celldisp(C)

D=ifo.dmats
celldisp(D)

A = [0]

B = [0]

C = [0]

D = [D{1,1}.*ones(ifo.outputsizes(1), ifo.inputsizes(1)  )
D{1,2}.*ones(ifo.outputsizes(1), ifo.inputsizes(2)  ) D{1,3}.*ones(ifo.outputsizes(1),
ifo.inputsizes(3)  ) D{1,4}.*ones(ifo.outputsizes(1), ifo.inputsizes(4)  );]

 %% LAPLACE

ifoSS = ss([],[],[],D)
[ifoSS,G] = c2d(ifoSS,0.1,'zoh')
tfifo = tf(ifoSS)

% per agafar entre input 1 i output 1

H = tfifo(1,1)
t = 0:0.1:1000-0.1;
u=sin(2*pi*0.002*t);

[y,t,x] = lsim(H,u,t)

%% SS

modifyTimeStep(ifo,0.1);
ifooutSSM = ifo.simulate(plist('AOS VARIABLE NAMES', {'EOM.tm1_x'}, 'AOS', input,'RETURN
OUTPUTS', {'IFO.x1'},'nsamples', 10000, 'fs', 10))
ifooutSSM = ifooutSSM.getObjectAtIndex(1);


%% figure

figure
plot(t, y, 'r')
hold on
plot(ifooutSSM.x, ifooutSSM.y, 'b')
legend('Ifo matlab', 'Ifo SSM')

figure
plot(t, (y - ifooutSSM.y) , 'r')
legend('diff')

%% ERROR

nsamples = 10000;

Abresult = 1/nsamples * sqrt( sum ( (y - ifooutSSM.y).^2  )  )

Relresult = 1/nsamples * sqrt( sum (y - ifooutSSM.y).^2/sum(ifooutSSM.y).^2)
```

LTP time-domain m-code

```
%% SSM

ltpda_startup;
%%
clear all

pl_w = plist('waveform', 'sine wave', 'phi', 0, 'f', 0.002, 'fs', 10, 'nsecs', 1000);
input = ao(pl_w);

ltpSSM = ssm(plist('built-in', 'ltp', 'dim', 1));


dfacs = ssm(plist('built-in', 'DFACS', 'dim', 1));
capact = ssm(plist('built-in', 'CAPACT', 'dim', 1));
accnoise = ssm(plist('built-in', 'ACCNOISE', 'dim', 1));
delay = ssm(plist('built-in', 'DELAY', 'dim', 1));


dfacs = modifyTimeStep(dfacs, 0.1);


%% FEEPS

feeps = ssm(plist('built-in', 'FEEPS', 'dim', 1));

Af=feeps.amats;
celldisp(Af);

Bf=feeps.bmats;
celldisp(Bf);

Cf=feeps.cmats;
celldisp(Cf);

Df=feeps.dmats;
celldisp(Df);

Af = [Af{1,1}.*ones(feeps.statesizes(1), feeps.statesizes(1)  );];

Bf = [Bf{1,1}.*ones(feeps.statesizes(1), feeps.inputsizes(1)  )
zeros(feeps.statesizes(1), feeps.inputsizes(2)  );];

Cf = [Cf{1,1}.*ones(feeps.outputsizes(1), feeps.statesizes(1)  );
     zeros(feeps.outputsizes(2), feeps.statesizes(1)  );];

Df = [Df{1,1}.*ones(feeps.outputsizes(1), feeps.inputsizes(1)  )
Df{1,2}.*ones(feeps.outputsizes(1), feeps.inputsizes(2)  );
     zeros(feeps.outputsizes(2), feeps.inputsizes(1)  )
Df{2,2}.*ones(feeps.outputsizes(2), feeps.inputsizes(2)  );];


feepsZOH = ss(Af,Bf,Cf,Df);
[feepsZOH,G] = c2d(feepsZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[0.2236]};
sys.bmats    = {[0.7764 0 0] [0]};
sys.cmats    = {[1] ; [0]};
sys.dmats    = {[0 0 0] [1] ;  [0 0 0] [1]};

sys.timestep = 0.1;

sys.name = 'FEEPSZOH';
sys.description = 'Feeps discretized with ZOH method';

inputnames    = {'DFACS' 'DIST_FEEPS'};
inputdescription = {'--' '--'};
inputvarnames = {{'sc_x' 'tm1_x' 'tm2_x'} {'sc_x'}};
inputvarunits = {[unit('kg m s^(-2)') unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg
m s^(-2)')]};
inputvardescription = [];

ssnames    = {'FEEPS'};
ssdescription = {'--'};
ssvarnames = {{'x'}};
ssvarunits={[unit('kg m s^(-1)')]};
ssvardescription = [];

outputnames    = {'FEEPS' 'FEEPSNOISE'};
outputdescription = {'--'};
```

```
outputvarnames ={{'x'} {'x'}};
outputvarunits={[unit('kg m s^-2')] [unit('kg m s^-2')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
feepsZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));


feepsFOH = ss(Af,Bf,Cf,Df);
[feepsFOH,G] = c2d(feepsFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[0.2236]};
sys.bmats    = {[0.4024 0 0] [0]};
sys.cmats    = {[1] ; [0]};
sys.dmats    = {[0.4817 0 0] [1] ;  [0 0 0] [1]};

sys.timestep = 0.1;

sys.name = 'FEEPSFOH';
sys.description = 'Feeps discretized with FOH method';

inputnames    = {'DFACS' 'DIST_FEEPS'};
inputdescription = {'--' '--'};
inputvarnames = {{'sc_x' 'tm1_x' 'tm2_x'} {'sc_x'}};
inputvarunits = {[unit('kg m s^(-2)') unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg
m s^(-2)')]};
inputvardescription = [];

ssnames    = {'FEEPS'};
ssdescription = {'--'};
ssvarnames = {{'x'}};
ssvarunits={[unit('kg m s^(-1)')]};
ssvardescription = [];

outputnames    = {'FEEPS' 'FEEPSNOISE'};
outputdescription = {'--'};
outputvarnames ={{'x'} {'x'}};
outputvarunits={[unit('kg m s^-2')] [unit('kg m s^-2')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
feepsFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

feepstustin = ss(Af,Bf,Cf,Df);
[feepstustin,G] = c2d(feepstustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[0.1436]};
sys.bmats    = {[8.564 0 0] [0]};
sys.cmats    = {[0.05718] ; [0]};
sys.dmats    = {[0.4282 0 0] [1] ;  [0 0 0] [1]};

sys.timestep = 0.1;

sys.name = 'FEEPStustin';
sys.description = 'Feeps discretized with Tustin method';
```

```
inputnames    = {'DFACS' 'DIST_FEEPS'};
inputdescription = {'--' '--'};
inputvarnames = {{'sc_x' 'tm1_x' 'tm2_x'} {'sc_x'}};
inputvarunits = {[unit('kg m s^(-2)') unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg
m s^(-2)')]};
inputvardescription = [];

ssnames    = {'FEEPS'};
ssdescription = {'--'};
ssvarnames = {{'x'}};
ssvarunits={[unit('kg m s^(-1)')]};
ssvardescription = [];

outputnames    = {'FEEPS' 'FEEPSNOISE'};
outputdescription = {'--'};
outputvarnames ={{'x'} {'x'}};
outputvarunits={[unit('kg m s^-2')] [unit('kg m s^-2')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
feepstustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% EOM

eom = ssm(plist('built-in', 'EOM', 'dim', 1));

Ae=eom.amats;
celldisp(Ae);

Be=eom.bmats;
celldisp(Be);

Ce=eom.cmats;
celldisp(Ce);

De=eom.dmats;
celldisp(De);

Ae = [                   0                          0                          1  0;
                         0                          0                          0  1;
      1.95397498328562e-06      9.67472981147119e-09                          0  0;
      6.50000000000002e-08                     2e-06                          0  0;];


Be = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      -0.00236574402649633 0.512662547978089 0.0024584663454362 -0.00236574402649633 -
0.00236574402649633 0.512662547978089 0.0024584663454362 -0.00236574402649633
0.510204081632653 0 -0.510204081632653 0.512662547978089 0.0024584663454362
0.510204081632653 0 0.510204081632653 0 0 0 0;
      0 -0.510204081632653 0.510204081632653 0 0 -0.510204081632653 0.510204081632653 0
-0.510204081632653 0.510204081632653 1.02040816326531 -0.510204081632653
0.510204081632653 -0.510204081632653 0.510204081632653 -0.510204081632653 -
0.510204081632653 0 0 0;];


Ce = [1 0 0 0;
      1 1 0 0;
      0 0 0 0;
      0 0 0 0;
      0 0 0 0;
      0 0 0 0;];

De = [De{1,1}.*ones(eom.outputsizes(1), eom.inputsizes(1)  ) zeros(eom.outputsizes(1),
eom.inputsizes(2)  ) zeros(eom.outputsizes(1), eom.inputsizes(3)  )
zeros(eom.outputsizes(1), eom.inputsizes(4)  ) zeros(eom.outputsizes(1),
eom.inputsizes(5)  ) zeros(eom.outputsizes(1), eom.inputsizes(6)  )
zeros(eom.outputsizes(1), eom.inputsizes(7)  ) zeros(eom.outputsizes(1),
eom.inputsizes(8)  ) zeros(eom.outputsizes(1), eom.inputsizes(9)  )
zeros(eom.outputsizes(1), eom.inputsizes(10)  ) zeros(eom.outputsizes(1),
eom.inputsizes(11)  ) zeros(eom.outputsizes(1), eom.inputsizes(12)  )
zeros(eom.outputsizes(1), eom.inputsizes(13)  );
      zeros(eom.outputsizes(2), eom.inputsizes(1)  ) De{2,2}.*ones(eom.outputsizes(2),
eom.inputsizes(2)  ) De{2,3}.*ones(eom.outputsizes(2), eom.inputsizes(3)  )
```

```
De{2,4}.*ones(eom.outputsizes(2), eom.inputsizes(4)  ) De{2,5}.*ones(eom.outputsizes(2),
eom.inputsizes(5)  ) De{2,6}.*ones(eom.outputsizes(2), eom.inputsizes(6)  )
De{2,7}.*ones(eom.outputsizes(2), eom.inputsizes(7)  ) De{2,8}.*ones(eom.outputsizes(2),
eom.inputsizes(8)  ) De{2,9}.*ones(eom.outputsizes(2), eom.inputsizes(9)  )
De{2,10}.*ones(eom.outputsizes(2), eom.inputsizes(10)  )
De{2,11}.*ones(eom.outputsizes(2), eom.inputsizes(11)  )
De{2,12}.*ones(eom.outputsizes(2), eom.inputsizes(12)  )
De{2,13}.*ones(eom.outputsizes(2), eom.inputsizes(13)  );
      De{3,1}.*ones(eom.outputsizes(3), eom.inputsizes(1)  )
De{3,2}.*ones(eom.outputsizes(3), eom.inputsizes(2)  ) De{3,3}.*ones(eom.outputsizes(3),
eom.inputsizes(3)  ) De{3,4}.*ones(eom.outputsizes(3), eom.inputsizes(4)  )
De{3,5}.*ones(eom.outputsizes(3), eom.inputsizes(5)  ) De{3,6}.*ones(eom.outputsizes(3),
eom.inputsizes(6)  ) De{3,7}.*ones(eom.outputsizes(3), eom.inputsizes(7)  )
De{3,8}.*ones(eom.outputsizes(3), eom.inputsizes(8)  ) De{3,9}.*ones(eom.outputsizes(3),
eom.inputsizes(9)  ) De{3,10}.*ones(eom.outputsizes(3), eom.inputsizes(10)  )
De{3,11}.*ones(eom.outputsizes(3), eom.inputsizes(11)  )
De{3,12}.*ones(eom.outputsizes(3), eom.inputsizes(12)  )
De{3,13}.*ones(eom.outputsizes(3), eom.inputsizes(13)  );];

eomZOH = ss(Ae,Be,Ce,De);
[eomZOH,G] = c2d(eomZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1 4.837e-11 0.1 1.612e-12;3.25e-10 1 1.083e-11 0.1;1.954e-07 9.675e-10
1 4.837e-11;6.5e-09 2e-07 3.25e-10 1]};
sys.bmats    = {[-1.183e-05;-6.407e-16;-0.0002366;-2.563e-14]    [0.002563   1.229e-05;-
0.002551    0.002551;0.05127   0.0002458;-0.05102     0.05102]  [-1.183e-05;-6.407e-16;-
0.0002366;-2.563e-14]  [-1.183e-05;-6.407e-16;-0.0002366;-2.563e-14]    [0.002563
1.229e-05;-0.002551    0.002551;0.05127   0.0002458;-0.05102     0.05102] [-1.183e-05;-
6.407e-16;-0.0002366;-2.563e-14]    [0.002551   2.057e-14;-0.002551    0.002551;0.05102
8.227e-13;-0.05102     0.05102]   [-0.002551;0.005102;-0.05102;0.102]    [0.002563
1.229e-05;-0.002551    0.002551;0.05127   0.0002458;-0.05102     0.05102]    [0.002551
2.057e-14;-0.002551 0.002551;0.05102 8.227e-13;-0.05102 0.05102]    [0.002551  -2.057e-
14;-0.002551   -0.002551;0.05102  -8.227e-13;-0.05102    -0.05102]           [0
0;0           0;0           0;0           0]           [0;0;0;0]};
sys.cmats    = {[1   0   0   0;1   1   0   0];
                [0   0   0   0;0   0   0   0];
                [0   0   0   0;0   0   0   0]};
sys.dmats    = {[0;0] [0 0;0 0] [0;0] [0;0] [0 0;0 0] [0;0] [0 0;0 0] [0;0] [0 0;0 0] [0
0;0 0]  [0 0;0 0]  [0 0;0 0]  [0;0];
                [0;0] [0 0;0 0] [-0.002366;0] [-0.002366;0] [0.5127 0.002458;-0.5102
0.5102] [-0.002366;0] [0.5102 0;-0.5102 0.5102] [-0.5102;1.02] [0.5127 0.002458;-0.5102
0.5102] [0.5102 0; -0.5102 0.5102] [0.5102 0;-0.5102 -0.5102] [0.5127 0.002458;-0.5102
0.5102]  [-0.002366;0];
                [-0.002366;0] [0.5127 0.002458;-0.5102 0.5102] [0;0] [0;0] [0 0;0 0]
[0;0] [0 0;0 0] [0;0] [0 0;0 0] [0 0;0 0] [0 0;0 0]  [0 0;0 0] [0;0]};

sys.timestep = 0.1;

sys.name = 'EOMZOH';
sys.description = 'Eom discretized with ZOH method';

inputnames    = {'FEEPS' 'CAPACT' 'DIST_SOLAR' 'DIST_INFRARED' 'DIST_TMSC' 'SIGNAL_SC'
'SIGNAL_TM' 'SIGNAL_TM12' 'SIGNAL_TMSC' 'DIST_TM' 'DIST_RIN' 'CAPACTNOISE'
'FEEPSNOISE'};
inputdescription = {'--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--'};
inputvarnames = {{'x'} {'tm1_x' 'tm2_x'} {'x'} {'x'} { 'tm1_x' 'tm2_x'} {'x'} {'tm1_x'
'tm2_x'} {'x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}
{'x'}};
inputvarunits = {[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')]};
inputvardescription = [];

ssnames    = {'EOM'};
ssdescription = {'--'};
ssvarnames = {{'tm1_x' 'tm12_x' 'tm1_xdot' 'tm12_xdot' }};
ssvarunits={[ unit('m') unit('m') unit('m s^(-1)') unit('m s^(-1)')]};
ssvardescription = [];

outputnames    = {'EOM' 'EOM_ACCNOISE' 'EOM_ACCCONTROL'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'} {'tm1_x' 'tm12_x'} {'tm1_x' 'tm12_x'}};
outputvarunits={[unit('m') unit('m')] [unit('m s^(-2)') unit('m s^(-2)')] [unit('m s^(-
2)') unit('m s^(-2)')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
```

```
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
eomZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));


eomFOH = ss(Ae,Be,Ce,De);
[eomFOH,G] = c2d(eomFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1 4.837e-11 0.1 1.612e-12;3.25e-10 1 1.083e-11 0.1;1.954e-07 9.675e-10
1 4.837e-11;6.5e-09 2e-07 3.25e-10 1]};

sys.bmats    = {[-2.366e-05;-3.844e-15;-0.0002366;-8.97e-14]     [0.005127   2.458e-05;-
0.005102    0.005102;0.05127   0.0002458;-0.05102     0.05102]  [-2.366e-05;-3.844e-15;-
0.0002366;-8.97e-14]  [-2.366e-05;-3.844e-15;-0.0002366;-8.97e-14]     [0.005127
2.458e-05;-0.005102    0.005102;0.05127   0.0002458;-0.05102     0.05102] [-2.366e-05;-
3.844e-15;-0.0002366;-8.97e-14]     [0.005102   1.234e-13;-0.005102    0.005102;0.05102
2.879e-12;-0.05102     0.05102]    [-0.005102;0.0102;-0.05102;0.102]     [0.005127
2.458e-05;-0.005102    0.005102;0.05127   0.0002458;-0.05102     0.05102]     [0.005102
1.234e-13;-0.005102 0.005102;0.05102 2.879e-12;-0.05102 0.05102]     [0.005102  -1.234e-
13;-0.005102   -0.005102;0.05102  -2.879e-12;-0.05102    -0.05102]            [0
0;0           0;0           0;0           0]           [0;0;0;0]};

sys.cmats    = {[1   0   0   0;1   1   0   0];
                [0   0   0   0;0   0   0   0];
                [0   0   0   0;0   0   0   0]};

sys.dmats    = {[-3.943e-06;-3.943e-06] [0.0008544   4.097e-06;4.097e-06   0.0008544] [-
3.943e-06;-3.943e-06] [-3.943e-06;-3.943e-06] [0.0008544   4.097e-06;4.097e-06
0.0008544] [-3.943e-06;-3.943e-06] [0.0008503   4.113e-15;3.954e-15   0.0008503] [-
0.0008503;0.0008503] [0.0008544   4.097e-06;4.097e-06   0.0008544] [0.0008503  4.113e-
15;3.954e-15  0.0008503] [0.0008503  -4.113e-15;3.954e-15  -0.0008503]  [0 0;0 0] [0;0];
                [0;0] [0 0;0 0] [-0.002366;0] [-0.002366;0] [0.5127 0.002458;-0.5102
0.5102] [-0.002366;0] [0.5102 0;-0.5102 0.5102] [-0.5102;1.02] [0.5127 0.002458;-0.5102
0.5102] [0.5102 0; -0.5102 0.5102] [0.5102 0;-0.5102 -0.5102] [0.5127 0.002458;-0.5102
0.5102]  [-0.002366;0];
                [-0.002366;0] [0.5127 0.002458;-0.5102 0.5102] [0;0] [0;0] [0 0;0 0]
[0;0] [0 0;0 0] [0;0] [0 0;0 0] [0 0;0 0] [0 0;0 0]  [0 0;0 0] [0;0]};

sys.timestep = 0.1;

sys.name = 'EOMFOH';
sys.description = 'Eom discretized with FOH method';

inputnames    = {'FEEPS' 'CAPACT' 'DIST_SOLAR' 'DIST_INFRARED' 'DIST_TMSC' 'SIGNAL_SC'
'SIGNAL_TM' 'SIGNAL_TM12' 'SIGNAL_TMSC' 'DIST_TM' 'DIST_RIN' 'CAPACTNOISE'
'FEEPSNOISE'};
inputdescription = {'--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--'};
inputvarnames = {{'x'} {'tm1_x' 'tm2_x'} {'x'} {'x'} { 'tm1_x' 'tm2_x'} {'x'} {'tm1_x'
'tm2_x'} {'x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}
{'x'}};
inputvarunits = {[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')]};
inputvardescription = [];

ssnames    = {'EOM'};
ssdescription = {'--'};
ssvarnames = {{'tm1_x' 'tm12_x' 'tm1_xdot' 'tm12_xdot' }};
ssvarunits={[ unit('m') unit('m') unit('m s^(-1)') unit('m s^(-1)')]};
ssvardescription = [];

outputnames    = {'EOM' 'EOM_ACCNOISE' 'EOM_ACCCONTROL'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'} {'tm1_x' 'tm12_x'} {'tm1_x' 'tm12_x'}};
outputvarunits={[unit('m') unit('m')] [unit('m s^(-2)') unit('m s^(-2)')] [unit('m s^(-
2)') unit('m s^(-2)')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
```

```
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
eomFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

eomtustin = ss(Ae,Be,Ce,De);
[eomtustin,G] = c2d(eomtustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1 4.837e-11 0.1 1.612e-12;3.25e-10 1 1.083e-11 0.1;1.954e-07 9.675e-10
1 4.837e-11;6.5e-09 2e-07 3.25e-10 1]};

sys.bmats    = {[-0.0001183;-1.922e-14;-0.002366;-3.844e-13]    [0.02563   0.0001229;-
0.02551     0.02551;0.5127    0.002458;-0.5102      0.5102]  [-0.0001183;-1.922e-14;-
0.002366;-3.844e-13]  [-0.0001183;-1.922e-14;-0.002366;-3.844e-13]    [0.02563
0.0001229;-0.02551     0.02551;0.5127    0.002458;-0.5102      0.5102] [-0.0001183;-
1.922e-14;-0.002366;-3.844e-13]    [0.02551    6.17e-13;-0.02551     0.02551;0.5102
1.234e-11;-0.5102      0.5102]   [-0.02551;0.05102;-0.5102;1.02]    [0.02563
0.0001229;-0.02551     0.02551;0.5127    0.002458;-0.5102      0.5102]    [0.02551
6.17e-13;-0.02551 0.02551;0.5102 1.234e-11;-0.5102 0.5102]    [0.02551   -6.17e-13;-
0.02551    -0.02551;0.5102  -1.234e-11;-0.5102     -0.5102]           [0           0;0
0;0           0;0           0]            [0;0;0;0]};

sys.cmats    = {[0.1  2.419e-12      0.005  1.209e-13;0.1        0.1      0.005
0.005];
                [0   0   0   0;0   0   0   0];
                [0   0   0   0;0   0   0   0]};

sys.dmats    = {[-5.914e-06;-5.914e-06] [0.001282   6.146e-06;6.146e-06    0.001282] [-
5.914e-06;-5.914e-06] [-5.914e-06;-5.914e-06] [0.001282   6.146e-06;6.146e-06
0.001282] [-5.914e-06;-5.914e-06] [0.001276   3.085e-14;2.966e-14    0.001276] [-
0.001276;0.001276] [0.001282   6.146e-06;6.146e-06    0.001282] [0.001276   3.085e-
14;2.966e-14   0.001276 ] [0.001276  -3.085e-14;2.966e-14   -0.001276]  [0 0;0 0] [0;0];
                [0;0] [0 0;0 0] [-0.002366;0] [-0.002366;0] [0.5127 0.002458;-0.5102
0.5102] [-0.002366;0] [0.5102 0;-0.5102 0.5102] [-0.5102;1.02] [0.5127 0.002458;-0.5102
0.5102] [0.5102 0; -0.5102 0.5102] [0.5102 0;-0.5102 -0.5102] [0.5127 0.002458;-0.5102
0.5102]  [-0.002366;0];
                [-0.002366;0] [0.5127 0.002458;-0.5102 0.5102] [0;0] [0;0] [0 0;0 0]
[0;0] [0 0;0 0] [0;0] [0 0;0 0] [0 0;0 0] [0 0;0 0]  [0 0;0 0] [0;0]};

sys.timestep = 0.1;

sys.name = 'EOMtustin';
sys.description = 'Eom discretized with Tustin method';

inputnames    = {'FEEPS' 'CAPACT' 'DIST_SOLAR' 'DIST_INFRARED' 'DIST_TMSC' 'SIGNAL_SC'
'SIGNAL_TM' 'SIGNAL_TM12' 'SIGNAL_TMSC' 'DIST_TM' 'DIST_RIN' 'CAPACTNOISE'
'FEEPSNOISE'};
inputdescription = {'--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--'};
inputvarnames = {{'x'} {'tm1_x' 'tm2_x'} {'x'} {'x'} { 'tm1_x' 'tm2_x'} {'x'} {'tm1_x'
'tm2_x'} {'x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}
{'x'}};
inputvarunits = {[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')]};
inputvardescription = [];

ssnames    = {'EOM'};
ssdescription = {'--'};
ssvarnames = {{'tm1_x' 'tm12_x' 'tm1_xdot' 'tm12_xdot'}};
ssvarunits={[ unit('m') unit('m') unit('m s^(-1)') unit('m s^(-1)')]};
ssvardescription = [];

outputnames    = {'EOM' 'EOM_ACCNOISE' 'EOM_ACCCONTROL'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'} {'tm1_x' 'tm12_x'} {'tm1_x' 'tm12_x'}};
outputvarunits={[unit('m') unit('m')] [unit('m s^(-2)') unit('m s^(-2)')] [unit('m s^(-
2)') unit('m s^(-2)')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
```

```
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
eomtustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));
%% IFO

ifo = ssm(plist('built-in', 'IFO', 'dim', 1));

Ai=ifo.amats;
celldisp(Ai);

Bi=ifo.bmats;
celldisp(Bi);

Ci=ifo.cmats;
celldisp(Ci);

Di=ifo.dmats;
celldisp(Di);

Ai = [0];

Bi = [0 0 0 0 0 0 0 0];

Ci = [0;
      0;];

Di = [1 0 1 0 1 0 1 0;
      -0.9999 1 0 1 -0.9999 1 0 1;];


ifoZOH = ss(Ai,Bi,Ci,Di);
[ifoZOH,G] = c2d(ifoZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0]   [0   0]  [0   0]   [0   0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;-0.9999 1] [1 0;0 1] [1 0;-0.9999 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IFOZOH';
sys.description = 'Ifo discretized with ZOH method';

inputnames    = {'EOM' 'DIST_IFO_READOUT' 'DIST_IFO_POSITION' 'SIGNAL_IFO'};
inputdescription = {'--' '--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'x1' 'x12'} {'x1' 'x12'} {'x1' 'x12'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]
[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IFO'};
outputdescription = {'--'};
outputvarnames ={'x1' 'x12'};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
ifoZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

ifoFOH = ss(Ai,Bi,Ci,Di);
```

```
[ifoFOH,G] = c2d(ifoFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0]   [0   0]  [0   0]   [0   0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;-0.9999 1] [1 0;0 1] [1 0;-0.9999 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IFOFOH';
sys.description = 'Ifo discretized with FOH method';

inputnames    = {'EOM' 'DIST_IFO_READOUT' 'DIST_IFO_POSITION' 'SIGNAL_IFO'};
inputdescription = {'--' '--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'x1' 'x12'} {'x1' 'x12'} {'x1' 'x12'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]
[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IFO'};
outputdescription = {'--'};
outputvarnames ={'x1' 'x12'};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
ifoFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

ifotustin = ss(Ai,Bi,Ci,Di);
[ifotustin,G] = c2d(ifotustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0]   [0   0]  [0   0]   [0   0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;-0.9999 1] [1 0;0 1] [1 0;-0.9999 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IFOtustin';
sys.description = 'Ifo discretized with Tustin method';

inputnames    = {'EOM' 'DIST_IFO_READOUT' 'DIST_IFO_POSITION' 'SIGNAL_IFO'};
inputdescription = {'--' '--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'x1' 'x12'} {'x1' 'x12'} {'x1' 'x12'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]
[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IFO'};
outputdescription = {'--'};
outputvarnames ={'x1' 'x12'};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
```

```
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
ifotustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% ST

st = ssm(plist('built-in', 'ST', 'dim', 1));

As=st.amats;
celldisp(As);

Bs=st.bmats;
celldisp(Bs);

Cs=st.cmats;
celldisp(Cs);

Ds=st.dmats;
celldisp(Ds);

As = [0];

Bs = [0 0 0];

Cs = [0];

Ds = [zeros(st.outputsizes(1), st.inputsizes(1)  ) zeros(st.outputsizes(1),
st.inputsizes(2)  ) zeros(st.outputsizes(1), st.inputsizes(3)  );];


stZOH = ss(As,Bs,Cs,Ds);
[stZOH,G] = c2d(stZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0]   [0]   [0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[0;0] [0;0] [0;0]};

sys.timestep = 0.1;

sys.name = 'STZOH';
sys.description = 'ST discretized with ZOH method';

inputnames    = {'EOM' 'DIST_ST' 'SIGNAL_ST'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = 0;
ssvarnames = 0;
ssvarunits= 0;
ssvardescription = 0;

outputnames    = {'ST'};
outputdescription = {'--'};
outputvarnames = {};
outputvarunits= {};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
stZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
```

```
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

stFOH = ss(As,Bs,Cs,Ds);
[stFOH,G] = c2d(stFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0   0]};
sys.cmats    = {[0]};
sys.dmats    = {[0 0 0]};

sys.timestep = 0.1;

sys.name = 'STFOH';
sys.description = 'ST discretized with FOH method';

inputnames    = {'EOM' 'DIST_ST' 'SIGNAL_ST'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = 0;
ssvarnames = 0;
ssvarunits= 0;
ssvardescription = 0;

outputnames    = {'ST'};
outputdescription = [];
outputvarnames = [];
outputvarunits= [];
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
stFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

sttustin = ss(As,Bs,Cs,Ds);
[sttustin,G] = c2d(sttustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0   0]};
sys.cmats    = {[0]};
sys.dmats    = {[0 0 0]};

sys.timestep = 0.1;

sys.name = 'STtustin';
sys.description = 'ST discretized with Tustin method';

inputnames    = {'EOM' 'DIST_ST' 'SIGNAL_ST'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = 0;
ssvarnames = 0;
ssvarunits= 0;
ssvardescription = 0;

outputnames    = {'ST'};
outputdescription = [];
outputvarnames = [];
outputvarunits= [];
outputvardescription = [];
```

```
% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
sttustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% IS

is = ssm(plist('built-in', 'IS', 'dim', 1));

Ais=is.amats;
celldisp(Ais);

Bis=is.bmats;
celldisp(Bis);

Cis=is.cmats;
celldisp(Cis);

Dis=is.dmats;
celldisp(Dis);

Ais = [0];

Bis = [0 0 0 0 0 0];

Cis = [0;
       0;];

Dis = [Dis{1,1}.*ones(is.outputsizes(1), is.inputsizes(1)  )
Dis{1,2}.*ones(is.outputsizes(1), is.inputsizes(2)  ) Dis{1,3}.*ones(is.outputsizes(1),
is.inputsizes(3)  );];

isZOH = ss(Ais,Bis,Cis,Dis);
[isZOH,G] = c2d(isZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0 0] [0 0] [0 0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;0 1] [1 0;0 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'ISZOH';
sys.description = 'IS discretized with ZOH method';

inputnames    = {'EOM' 'DIST_IS' 'SIGNAL_IS'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IS'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'}};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
```

```
isZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

isFOH = ss(Ais,Bis,Cis,Dis);
[isFOH,G] = c2d(isFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0 0] [0 0] [0 0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;0 1] [1 0;0 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'ISFOH';
sys.description = 'IS discretized with FOH method';

inputnames    = {'EOM' 'DIST_IS' 'SIGNAL_IS'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IS'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'}};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
isFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

istustin = ss(Ais,Bis,Cis,Dis);
[istustin,G] = c2d(istustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0 0] [0 0] [0 0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;0 1] [1 0;0 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IStustin';
sys.description = 'IS discretized with Tustin method';

inputnames    = {'EOM' 'DIST_IS' 'SIGNAL_IS'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IS'};
outputdescription = {'--'};
```

```
outputvarnames ={{'tm1_x' 'tm2_x'}};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
istustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% assembling


ltpZOH = assemble([feepsZOH eomZOH ifoZOH isZOH delay dfacs capact accnoise])
ltpFOH = assemble([feepsFOH eomFOH ifoFOH isFOH delay dfacs capact accnoise]);
ltpTustin = assemble([feepstustin eomtustin ifotustin istustin delay dfacs capact
accnoise]);

t = 0:0.1:1000-0.1;

%% SS

ltpoutSSM = ltpSSM.simulate(plist('AOS VARIABLE NAMES', {'DIST_FEEPS.sc_x'}, 'AOS',
input,'RETURN OUTPUTS', {'EOM.tm1_x'},'nsamples', 10000, 'fs', 10));
ltpoutSSM = ltpoutSSM.getObjectAtIndex(1);

ltpoutZOH = ltpZOH.simulate(plist('AOS VARIABLE NAMES', {'DIST_FEEPS.sc_x'}, 'AOS',
input,'RETURN OUTPUTS', {'EOM.tm1_x'},'nsamples', 10000, 'fs', 10));
ltpoutZOH = ltpoutZOH.getObjectAtIndex(1);

ltpoutFOH = ltpFOH.simulate(plist('AOS VARIABLE NAMES', {'DIST_FEEPS.sc_x'}, 'AOS',
input,'RETURN OUTPUTS', {'EOM.tm1_x'},'nsamples', 10000, 'fs', 10));
ltpoutFOH = ltpoutFOH.getObjectAtIndex(1);

ltpoutTustin = ltpTustin.simulate(plist('AOS VARIABLE NAMES', {'DIST_FEEPS.sc_x'},
'AOS', input,'RETURN OUTPUTS', {'EOM.tm1_x'},'nsamples', 10000, 'fs', 10));
ltpoutTustin = ltpoutTustin.getObjectAtIndex(1);

%% figure


figure
plot(ltpoutZOH.x, ltpoutZOH.y, 'r')
hold on
plot(ltpoutSSM.x, ltpoutSSM.y, 'b')
legend('LTP ZOH''LTP SSM')


figure
plot(ltpoutFOH.x, ltpoutFOH.y, 'r')
hold on
plot(ltpoutSSM.x, ltpoutSSM.y, 'b')
legend('LTP FOH''LTP SSM')


figure
plot(ltpoutTustin.x, ltpoutTustin.y, 'r')
hold on
plot(ltpoutSSM.x, ltpoutSSM.y, 'b')
legend('LTP Tustin''LTP SSM')



figure
plot(t, (ltpoutZOH.y - ltpoutSSM.y) , 'r')
legend('diff')

figure
plot(t, (ltpoutFOH.y - ltpoutSSM.y) , 'r')
legend('diff')

figure
plot(t, (ltpoutTustin.y - ltpoutSSM.y) , 'r')
legend('diff')

%% ERROR

nsamples = 10000;
```

```
AbsZ = 1/nsamples * sqrt( sum ( (ltpoutZOH.y - ltpoutSSM.y).^2  )  )
RelZ = 1/nsamples * sqrt( sum (ltpoutZOH.y - ltpoutSSM.y).^2/sum(ltpoutSSM.y).^2)

AbsF = 1/nsamples * sqrt( sum ( (ltpoutFOH.y - ltpoutSSM.y).^2  )  )
RelF = 1/nsamples * sqrt( sum (ltpoutFOH.y - ltpoutSSM.y).^2/sum(ltpoutSSM.y).^2)

AbsT = 1/nsamples * sqrt( sum ( (ltpoutTustin.y - ltpoutSSM.y).^2  )  )
RelT = 1/nsamples * sqrt( sum (ltpoutTustin.y - ltpoutSSM.y).^2/sum(ltpoutSSM.y).^2)
```

FEEPS frequency-domain m-code

```
%% SSM

ltpda_startup;
%%
clear all

f = logspace(-3, 0, 1000);



feeps = ssm(plist('built-in', 'FEEPS', 'dim', 1));


%% CONTINUOUS MATRICES

A=feeps.amats
celldisp(A)

B=feeps.bmats
celldisp(B)

C=feeps.cmats
celldisp(C)

D=feeps.dmats
celldisp(D)

A = [A{1,1}.*ones(feeps.statesizes(1), feeps.statesizes(1)  );]

B = [B{1,1}.*ones(feeps.statesizes(1), feeps.inputsizes(1)  ) zeros(feeps.statesizes(1),
feeps.inputsizes(2)  );]

C = [C{1,1}.*ones(feeps.outputsizes(1), feeps.statesizes(1)  );
     zeros(feeps.outputsizes(2), feeps.statesizes(1)  );]

D = [D{1,1}.*ones(feeps.outputsizes(1), feeps.inputsizes(1)  )
D{1,2}.*ones(feeps.outputsizes(1), feeps.inputsizes(2)  );
     zeros(feeps.outputsizes(2), feeps.inputsizes(1)  )
D{2,2}.*ones(feeps.outputsizes(2), feeps.inputsizes(2)  );]

 %% LAPLACE

feepsSS = ss(A,B,C,D)
[feepsSS,G] = c2d(feepsSS,0.1,'tustin')
tffeeps = tf(feepsSS)

% per agafar entre input 1 i output 1

H = tffeeps(1,1)
[mag, phase, w] = bode(H, 2*pi*f);

%% SS

modifyTimeStep(feeps,0.1);
feepsoutSSM = bode(feeps, plist('f',f, 'inputs', {'DFACS.sc_x'}, 'outputs',
{'FEEPS.x'}))
feepsoutSSM = feepsoutSSM.getObjectAtIndex(1);


%% figure

figure
semilogx(w/2/pi, squeeze(abs(mag)), 'r')
hold on
semilogx(feepsoutSSM.x, abs(feepsoutSSM.y), 'b')
legend('Feeps matlab', 'Feeps SSM')


figure
loglog(w/2/pi, abs(squeeze(abs(mag)) - abs(feepsoutSSM.y)) , 'r')
legend('diff')


%% ERROR

nsamples = 1000;

Abresult = 1/nsamples * sqrt( sum ( (abs(squeeze(abs(mag)) - abs(feepsoutSSM.y))).^2  )
)

Relresult = 1/nsamples * sqrt( sum(abs(squeeze(abs(mag)) -
abs(feepsoutSSM.y))).^2/sum(abs(feepsoutSSM.y).^2))
```

IFO frequency-domain m-code

```
%% SSM

ltpda_startup;
%%
clear all

f = logspace(-3, 0, 1000);

ifo = ssm(plist('built-in', 'IFO', 'dim', 1));

%% MATRICES

A=ifo.amats
celldisp(A)

B=ifo.bmats
celldisp(B)

C=ifo.cmats
celldisp(C)

D=ifo.dmats
celldisp(D)

A = [0]

B = [0]

C = [0]

D = [D{1,1}.*ones(ifo.outputsizes(1), ifo.inputsizes(1)  )
D{1,2}.*ones(ifo.outputsizes(1), ifo.inputsizes(2)  ) D{1,3}.*ones(ifo.outputsizes(1),
ifo.inputsizes(3)  ) D{1,4}.*ones(ifo.outputsizes(1), ifo.inputsizes(4)  );]

 %% LAPLACE

ifoSS = ss([],[],[],D)
[ifoSS,G] = c2d(ifoSS,0.1,'foh')
tfifo = tf(ifoSS)

% per agafar entre input 1 i output 1

H = tfifo(1,1)
[mag, phase, w] = bode(H, 2*pi*f);

%% SS

modifyTimeStep(ifo,0.1);
ifooutSSM = bode(ifo, plist('f',f, 'inputs', {'EOM.tm1_x'}, 'outputs', {'IFO.x1'}))
ifooutSSM = ifooutSSM.getObjectAtIndex(1);


%% figure

figure
semilogx(w/2/pi, squeeze(abs(mag)), 'r')
hold on
semilogx(ifooutSSM.x, abs(ifooutSSM.y), 'b')
legend('Ifo matlab', 'Ifo SSM')

figure
plot(w/2/pi, abs((squeeze(abs(mag)) - abs(ifooutSSM.y))) , 'r')
legend('diff')

%% ERROR

nsamples = 1000;

result = 1/nsamples * sqrt( sum ( (squeeze(abs(mag)) - abs(ifooutSSM.y)).^2  )  )
```

LTP fequency-domain m-code

```
%% SSM

ltpda_startup;
%%
clear all

f = logspace(-3, 2, 1000);

ltpSSM = ssm(plist('built-in', 'ltp', 'dim', 1));


dfacs = ssm(plist('built-in', 'DFACS', 'dim', 1));
capact = ssm(plist('built-in', 'CAPACT', 'dim', 1));
accnoise = ssm(plist('built-in', 'ACCNOISE', 'dim', 1));
delay = ssm(plist('built-in', 'DELAY', 'dim', 1));


dfacs = modifyTimeStep(dfacs, 0.1);


%% FEEPS

feeps = ssm(plist('built-in', 'FEEPS', 'dim', 1));

Af=feeps.amats;
celldisp(Af);

Bf=feeps.bmats;
celldisp(Bf);

Cf=feeps.cmats;
celldisp(Cf);

Df=feeps.dmats;
celldisp(Df);

Af = [Af{1,1}.*ones(feeps.statesizes(1), feeps.statesizes(1)  );];

Bf = [Bf{1,1}.*ones(feeps.statesizes(1), feeps.inputsizes(1)  )
zeros(feeps.statesizes(1), feeps.inputsizes(2)  );];

Cf = [Cf{1,1}.*ones(feeps.outputsizes(1), feeps.statesizes(1)  );
     zeros(feeps.outputsizes(2), feeps.statesizes(1)  );];

Df = [Df{1,1}.*ones(feeps.outputsizes(1), feeps.inputsizes(1)  )
Df{1,2}.*ones(feeps.outputsizes(1), feeps.inputsizes(2)  );
     zeros(feeps.outputsizes(2), feeps.inputsizes(1)  )
Df{2,2}.*ones(feeps.outputsizes(2), feeps.inputsizes(2)  );];


feepsZOH = ss(Af,Bf,Cf,Df);
[feepsZOH,G] = c2d(feepsZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[0.2236]};
sys.bmats    = {[0.7764 0 0] [0]};
sys.cmats    = {[1] ; [0]};
sys.dmats    = {[0 0 0] [1] ;  [0 0 0] [1]};

sys.timestep = 0.1;

sys.name = 'FEEPSZOH';
sys.description = 'Feeps discretized with ZOH method';

inputnames    = {'DFACS' 'DIST_FEEPS'};
inputdescription = {'--' '--'};
inputvarnames = {{'sc_x' 'tm1_x' 'tm2_x'} {'sc_x'}};
inputvarunits = {[unit('kg m s^(-2)') unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg
m s^(-2)')]};
inputvardescription = [];

ssnames    = {'FEEPS'};
ssdescription = {'--'};
ssvarnames = {{'x'}};
ssvarunits={[unit('kg m s^(-1)')]};
ssvardescription = [];

outputnames    = {'FEEPS' 'FEEPSNOISE'};
```

```
outputdescription = {'--'};
outputvarnames ={{'x'} {'x'}};
outputvarunits={[unit('kg m s^-2')] [unit('kg m s^-2')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
feepsZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));


feepsFOH = ss(Af,Bf,Cf,Df);
[feepsFOH,G] = c2d(feepsFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[0.2236]};
sys.bmats    = {[0.4024 0 0] [0]};
sys.cmats    = {[1] ; [0]};
sys.dmats    = {[0.4817 0 0] [1] ;  [0 0 0] [1]};

sys.timestep = 0.1;

sys.name = 'FEEPSFOH';
sys.description = 'Feeps discretized with FOH method';

inputnames    = {'DFACS' 'DIST_FEEPS'};
inputdescription = {'--' '--'};
inputvarnames = {{'sc_x' 'tm1_x' 'tm2_x'} {'sc_x'}};
inputvarunits = {[unit('kg m s^(-2)') unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg
m s^(-2)')]};
inputvardescription = [];

ssnames    = {'FEEPS'};
ssdescription = {'--'};
ssvarnames = {{'x'}};
ssvarunits={[unit('kg m s^(-1)')]};
ssvardescription = [];

outputnames    = {'FEEPS' 'FEEPSNOISE'};
outputdescription = {'--'};
outputvarnames ={{'x'} {'x'}};
outputvarunits={[unit('kg m s^-2')] [unit('kg m s^-2')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
feepsFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

feepstustin = ss(Af,Bf,Cf,Df);
[feepstustin,G] = c2d(feepstustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[0.1436]};
sys.bmats    = {[8.564 0 0] [0]};
sys.cmats    = {[0.05718] ; [0]};
sys.dmats    = {[0.4282 0 0] [1] ;  [0 0 0] [1]};

sys.timestep = 0.1;

sys.name = 'FEEPStustin';
```

```
sys.description = 'Feeps discretized with Tustin method';

inputnames    = {'DFACS' 'DIST_FEEPS'};
inputdescription = {'--' '--'};
inputvarnames = {{'sc_x' 'tm1_x' 'tm2_x'} {'sc_x'}};
inputvarunits = {[unit('kg m s^(-2)') unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg
m s^(-2)')]};
inputvardescription = [];

ssnames    = {'FEEPS'};
ssdescription = {'--'};
ssvarnames = {{'x'}};
ssvarunits={[unit('kg m s^(-1)')]};
ssvardescription = [];

outputnames    = {'FEEPS' 'FEEPSNOISE'};
outputdescription = {'--'};
outputvarnames ={{'x'} {'x'}};
outputvarunits={[unit('kg m s^-2')] [unit('kg m s^-2')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
feepstustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% EOM

eom = ssm(plist('built-in', 'EOM', 'dim', 1));

Ae=eom.amats;
celldisp(Ae);

Be=eom.bmats;
celldisp(Be);

Ce=eom.cmats;
celldisp(Ce);

De=eom.dmats;
celldisp(De);

Ae = [                   0                          0                          1  0;
                         0                          0                          0  1;
      1.95397498328562e-06      9.67472981147119e-09                          0  0;
      6.50000000000002e-08                      2e-06                          0  0;];


Be = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
      -0.00236574402649633 0.512662547978089 0.0024584663454362 -0.00236574402649633 -
0.00236574402649633 0.512662547978089 0.0024584663454362 -0.00236574402649633
0.510204081632653 0 -0.510204081632653 0.512662547978089 0.0024584663454362
0.510204081632653 0 0.510204081632653 0 0 0 0;
      0 -0.510204081632653 0.510204081632653 0 0 -0.510204081632653 0.510204081632653 0
-0.510204081632653 0.510204081632653 1.02040816326531 -0.510204081632653
0.510204081632653 -0.510204081632653 0.510204081632653 -0.510204081632653 -
0.510204081632653 0 0 0;];


Ce = [1 0 0 0;
      1 1 0 0;
      0 0 0 0;
      0 0 0 0;
      0 0 0 0;
      0 0 0 0;];

De = [De{1,1}.*ones(eom.outputsizes(1), eom.inputsizes(1)  ) zeros(eom.outputsizes(1),
eom.inputsizes(2)  ) zeros(eom.outputsizes(1), eom.inputsizes(3)  )
zeros(eom.outputsizes(1), eom.inputsizes(4)  ) zeros(eom.outputsizes(1),
eom.inputsizes(5)  ) zeros(eom.outputsizes(1), eom.inputsizes(6)  )
zeros(eom.outputsizes(1), eom.inputsizes(7)  ) zeros(eom.outputsizes(1),
eom.inputsizes(8)  ) zeros(eom.outputsizes(1), eom.inputsizes(9)  )
zeros(eom.outputsizes(1), eom.inputsizes(10)  ) zeros(eom.outputsizes(1),
eom.inputsizes(11)  ) zeros(eom.outputsizes(1), eom.inputsizes(12)  )
zeros(eom.outputsizes(1), eom.inputsizes(13)  );
```

```
      zeros(eom.outputsizes(2), eom.inputsizes(1)  ) De{2,2}.*ones(eom.outputsizes(2),
eom.inputsizes(2)  ) De{2,3}.*ones(eom.outputsizes(2), eom.inputsizes(3)  )
De{2,4}.*ones(eom.outputsizes(2), eom.inputsizes(4)  ) De{2,5}.*ones(eom.outputsizes(2),
eom.inputsizes(5)  ) De{2,6}.*ones(eom.outputsizes(2), eom.inputsizes(6)  )
De{2,7}.*ones(eom.outputsizes(2), eom.inputsizes(7)  ) De{2,8}.*ones(eom.outputsizes(2),
eom.inputsizes(8)  ) De{2,9}.*ones(eom.outputsizes(2), eom.inputsizes(9)  )
De{2,10}.*ones(eom.outputsizes(2), eom.inputsizes(10)  )
De{2,11}.*ones(eom.outputsizes(2), eom.inputsizes(11)  )
De{2,12}.*ones(eom.outputsizes(2), eom.inputsizes(12)  )
De{2,13}.*ones(eom.outputsizes(2), eom.inputsizes(13)  );
      De{3,1}.*ones(eom.outputsizes(3), eom.inputsizes(1)  )
De{3,2}.*ones(eom.outputsizes(3), eom.inputsizes(2)  ) De{3,3}.*ones(eom.outputsizes(3),
eom.inputsizes(3)  ) De{3,4}.*ones(eom.outputsizes(3), eom.inputsizes(4)  )
De{3,5}.*ones(eom.outputsizes(3), eom.inputsizes(5)  ) De{3,6}.*ones(eom.outputsizes(3),
eom.inputsizes(6)  ) De{3,7}.*ones(eom.outputsizes(3), eom.inputsizes(7)  )
De{3,8}.*ones(eom.outputsizes(3), eom.inputsizes(8)  ) De{3,9}.*ones(eom.outputsizes(3),
eom.inputsizes(9)  ) De{3,10}.*ones(eom.outputsizes(3), eom.inputsizes(10)  )
De{3,11}.*ones(eom.outputsizes(3), eom.inputsizes(11)  )
De{3,12}.*ones(eom.outputsizes(3), eom.inputsizes(12)  )
De{3,13}.*ones(eom.outputsizes(3), eom.inputsizes(13)  );];

eomZOH = ss(Ae,Be,Ce,De);
[eomZOH,G] = c2d(eomZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1 4.837e-11 0.1 1.612e-12;3.25e-10 1 1.083e-11 0.1;1.954e-07 9.675e-10
1 4.837e-11;6.5e-09 2e-07 3.25e-10 1]};
sys.bmats    = {[-1.183e-05;-6.407e-16;-0.0002366;-2.563e-14]    [0.002563   1.229e-05;-
0.002551    0.002551;0.05127   0.0002458;-0.05102     0.05102]  [-1.183e-05;-6.407e-16;-
0.0002366;-2.563e-14]  [-1.183e-05;-6.407e-16;-0.0002366;-2.563e-14]    [0.002563
1.229e-05;-0.002551    0.002551;0.05127   0.0002458;-0.05102     0.05102] [-1.183e-05;-
6.407e-16;-0.0002366;-2.563e-14]    [0.002551   2.057e-14;-0.002551    0.002551;0.05102
8.227e-13;-0.05102     0.05102]   [-0.002551;0.005102;-0.05102;0.102]    [0.002563
1.229e-05;-0.002551    0.002551;0.05127   0.0002458;-0.05102     0.05102]    [0.002551
2.057e-14;-0.002551 0.002551;0.05102 8.227e-13;-0.05102 0.05102]    [0.002551  -2.057e-
14;-0.002551   -0.002551;0.05102  -8.227e-13;-0.05102    -0.05102]           [0
0;0           0;0           0;0           0]           [0;0;0;0]};
sys.cmats    = {[1   0   0   0;1   1   0   0];
                [0   0   0   0;0   0   0   0];
                [0   0   0   0;0   0   0   0]};
sys.dmats    = {[0;0] [0 0;0 0] [0;0] [0;0] [0 0;0 0] [0;0] [0 0;0 0] [0;0] [0 0;0 0] [0
0;0 0] [0 0;0 0]  [0 0;0 0] [0;0];
                [0;0] [0 0;0 0] [-0.002366;0] [-0.002366;0] [0.5127 0.002458;-0.5102
0.5102] [-0.002366;0] [0.5102 0;-0.5102 0.5102] [-0.5102;1.02] [0.5127 0.002458;-0.5102
0.5102] [0.5102 0; -0.5102 0.5102] [0.5102 0;-0.5102 -0.5102] [0.5127 0.002458;-0.5102
0.5102]  [-0.002366;0];
                [-0.002366;0] [0.5127 0.002458;-0.5102 0.5102] [0;0] [0;0] [0 0;0 0]
[0;0] [0 0;0 0] [0;0] [0 0;0 0] [0 0;0 0] [0 0;0 0]  [0 0;0 0] [0;0]};

sys.timestep = 0.1;

sys.name = 'EOMZOH';
sys.description = 'Eom discretized with ZOH method';

inputnames    = {'FEEPS' 'CAPACT' 'DIST_SOLAR' 'DIST_INFRARED' 'DIST_TMSC' 'SIGNAL_SC'
'SIGNAL_TM' 'SIGNAL_TM12' 'SIGNAL_TMSC' 'DIST_TM' 'DIST_RIN' 'CAPACTNOISE'
'FEEPSNOISE'};
inputdescription = {'--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--'};
inputvarnames = {{'x'} {'tm1_x' 'tm2_x'} {'x'} {'x'} { 'tm1_x' 'tm2_x'} {'x'} {'tm1_x'
'tm2_x'} {'x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}
{'x'}};
inputvarunits = {[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')]};
inputvardescription = [];

ssnames    = {'EOM'};
ssdescription = {'--'};
ssvarnames = {{'tm1_x' 'tm12_x' 'tm1_xdot' 'tm12_xdot' }};
ssvarunits={[ unit('m') unit('m') unit('m s^(-1)') unit('m s^(-1)')]};
ssvardescription = [];

outputnames    = {'EOM' 'EOM_ACCNOISE' 'EOM_ACCCONTROL'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'} {'tm1_x' 'tm12_x'} {'tm1_x' 'tm12_x'}};
outputvarunits={[unit('m') unit('m')] [unit('m s^(-2)') unit('m s^(-2)')] [unit('m s^(-
2)') unit('m s^(-2)')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
```

```
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
eomZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));


eomFOH = ss(Ae,Be,Ce,De);
[eomFOH,G] = c2d(eomFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1 4.837e-11 0.1 1.612e-12;3.25e-10 1 1.083e-11 0.1;1.954e-07 9.675e-10
1 4.837e-11;6.5e-09 2e-07 3.25e-10 1]};

sys.bmats    = {[-2.366e-05;-3.844e-15;-0.0002366;-8.97e-14]    [0.005127   2.458e-05;-
0.005102    0.005102;0.05127   0.0002458;-0.05102     0.05102]  [-2.366e-05;-3.844e-15;-
0.0002366;-8.97e-14]  [-2.366e-05;-3.844e-15;-0.0002366;-8.97e-14]    [0.005127
2.458e-05;-0.005102    0.005102;0.05127   0.0002458;-0.05102     0.05102] [-2.366e-05;-
3.844e-15;-0.0002366;-8.97e-14]    [0.005102   1.234e-13;-0.005102    0.005102;0.05102
2.879e-12;-0.05102     0.05102]   [-0.005102;0.0102;-0.05102;0.102]    [0.005127
2.458e-05;-0.005102    0.005102;0.05127   0.0002458;-0.05102     0.05102]    [0.005102
1.234e-13;-0.005102 0.005102;0.05102 2.879e-12;-0.05102 0.05102]    [0.005102  -1.234e-
13;-0.005102   -0.005102;0.05102  -2.879e-12;-0.05102    -0.05102]           [0
0;0           0;0           0;0           0]           [0;0;0;0]};

sys.cmats    = {[1   0   0   0;1   1   0   0];
                [0   0   0   0;0   0   0   0];
                [0   0   0   0;0   0   0   0]};

sys.dmats    = {[-3.943e-06;-3.943e-06] [0.0008544   4.097e-06;4.097e-06   0.0008544] [-
3.943e-06;-3.943e-06] [-3.943e-06;-3.943e-06] [0.0008544   4.097e-06;4.097e-06
0.0008544] [-3.943e-06;-3.943e-06] [0.0008503   4.113e-15;3.954e-15   0.0008503] [-
0.0008503;0.0008503] [0.0008544   4.097e-06;4.097e-06   0.0008544] [0.0008503  4.113e-
15;3.954e-15  0.0008503] [0.0008503  -4.113e-15;3.954e-15  -0.0008503]  [0 0;0 0] [0;0];
                [0;0] [0 0;0 0] [-0.002366;0] [-0.002366;0] [0.5127 0.002458;-0.5102
0.5102] [-0.002366;0] [0.5102 0;-0.5102 0.5102] [-0.5102;1.02] [0.5127 0.002458;-0.5102
0.5102] [0.5102 0; -0.5102 0.5102] [0.5102 0;-0.5102 -0.5102] [0.5127 0.002458;-0.5102
0.5102]  [-0.002366;0];
                [-0.002366;0] [0.5127 0.002458;-0.5102 0.5102] [0;0] [0;0] [0 0;0 0]
[0;0] [0 0;0 0] [0;0] [0 0;0 0] [0 0;0 0] [0 0;0 0]  [0 0;0 0] [0;0]};

sys.timestep = 0.1;

sys.name = 'EOMFOH';
sys.description = 'Eom discretized with FOH method';

inputnames    = {'FEEPS' 'CAPACT' 'DIST_SOLAR' 'DIST_INFRARED' 'DIST_TMSC' 'SIGNAL_SC'
'SIGNAL_TM' 'SIGNAL_TM12' 'SIGNAL_TMSC' 'DIST_TM' 'DIST_RIN' 'CAPACTNOISE'
'FEEPSNOISE'};
inputdescription = {'--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--'};
inputvarnames = {{'x'} {'tm1_x' 'tm2_x'} {'x'} {'x'} { 'tm1_x' 'tm2_x'} {'x'} {'tm1_x'
'tm2_x'} {'x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}
{'x'}};
inputvarunits = {[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')]};
inputvardescription = [];

ssnames    = {'EOM'};
ssdescription = {'--'};
ssvarnames = {{'tm1_x' 'tm12_x' 'tm1_xdot' 'tm12_xdot' }};
ssvarunits={[ unit('m') unit('m') unit('m s^(-1)') unit('m s^(-1)')]};
ssvardescription = [];

outputnames    = {'EOM' 'EOM_ACCNOISE' 'EOM_ACCCONTROL'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'} {'tm1_x' 'tm12_x'} {'tm1_x' 'tm12_x'}};
outputvarunits={[unit('m') unit('m')] [unit('m s^(-2)') unit('m s^(-2)')] [unit('m s^(-
2)') unit('m s^(-2)')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
```

```
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
eomFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

eomtustin = ss(Ae,Be,Ce,De);
[eomtustin,G] = c2d(eomtustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1 4.837e-11 0.1 1.612e-12;3.25e-10 1 1.083e-11 0.1;1.954e-07 9.675e-10
1 4.837e-11;6.5e-09 2e-07 3.25e-10 1]};

sys.bmats    = {[-0.0001183;-1.922e-14;-0.002366;-3.844e-13]    [0.02563   0.0001229;-
0.02551     0.02551;0.5127    0.002458;-0.5102      0.5102]  [-0.0001183;-1.922e-14;-
0.002366;-3.844e-13]  [-0.0001183;-1.922e-14;-0.002366;-3.844e-13]    [0.02563
0.0001229;-0.02551     0.02551;0.5127    0.002458;-0.5102      0.5102] [-0.0001183;-
1.922e-14;-0.002366;-3.844e-13]    [0.02551    6.17e-13;-0.02551     0.02551;0.5102
1.234e-11;-0.5102      0.5102]   [-0.02551;0.05102;-0.5102;1.02]    [0.02563
0.0001229;-0.02551     0.02551;0.5127    0.002458;-0.5102      0.5102]    [0.02551
6.17e-13;-0.02551 0.02551;0.5102 1.234e-11;-0.5102 0.5102]    [0.02551   -6.17e-13;-
0.02551    -0.02551;0.5102  -1.234e-11;-0.5102     -0.5102]           [0           0;0
0;0           0;0           0]            [0;0;0;0]};

sys.cmats    = {[0.1  2.419e-12      0.005  1.209e-13;0.1        0.1      0.005
0.005];
                [0   0   0   0;0   0   0   0];
                [0   0   0   0;0   0   0   0]};

sys.dmats    = {[-5.914e-06;-5.914e-06] [0.001282   6.146e-06;6.146e-06    0.001282] [-
5.914e-06;-5.914e-06] [-5.914e-06;-5.914e-06] [0.001282   6.146e-06;6.146e-06
0.001282] [-5.914e-06;-5.914e-06] [0.001276   3.085e-14;2.966e-14    0.001276] [-
0.001276;0.001276] [0.001282   6.146e-06;6.146e-06    0.001282] [0.001276   3.085e-
14;2.966e-14   0.001276 ] [0.001276  -3.085e-14;2.966e-14   -0.001276]  [0 0;0 0] [0;0];
                [0;0] [0 0;0 0] [-0.002366;0] [-0.002366;0] [0.5127 0.002458;-0.5102
0.5102] [-0.002366;0] [0.5102 0;-0.5102 0.5102] [-0.5102;1.02] [0.5127 0.002458;-0.5102
0.5102] [0.5102 0; -0.5102 0.5102] [0.5102 0;-0.5102 -0.5102] [0.5127 0.002458;-0.5102
0.5102]  [-0.002366;0];
                [-0.002366;0] [0.5127 0.002458;-0.5102 0.5102] [0;0] [0;0] [0 0;0 0]
[0;0] [0 0;0 0] [0;0] [0 0;0 0] [0 0;0 0] [0 0;0 0]  [0 0;0 0] [0;0]};

sys.timestep = 0.1;

sys.name = 'EOMtustin';
sys.description = 'Eom discretized with Tustin method';

inputnames    = {'FEEPS' 'CAPACT' 'DIST_SOLAR' 'DIST_INFRARED' 'DIST_TMSC' 'SIGNAL_SC'
'SIGNAL_TM' 'SIGNAL_TM12' 'SIGNAL_TMSC' 'DIST_TM' 'DIST_RIN' 'CAPACTNOISE'
'FEEPSNOISE'};
inputdescription = {'--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--' '--'};
inputvarnames = {{'x'} {'tm1_x' 'tm2_x'} {'x'} {'x'} { 'tm1_x' 'tm2_x'} {'x'} {'tm1_x'
'tm2_x'} {'x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}
{'x'}};
inputvarunits = {[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)') unit('kg m s^(-2)')] [unit('kg m s^(-2)') unit('kg m s^(-2)')]
[unit('kg m s^(-2)')]};
inputvardescription = [];

ssnames    = {'EOM'};
ssdescription = {'--'};
ssvarnames = {{'tm1_x' 'tm12_x' 'tm1_xdot' 'tm12_xdot'}};
ssvarunits={[ unit('m') unit('m') unit('m s^(-1)') unit('m s^(-1)')]};
ssvardescription = [];

outputnames    = {'EOM' 'EOM_ACCNOISE' 'EOM_ACCCONTROL'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'} {'tm1_x' 'tm12_x'} {'tm1_x' 'tm12_x'}};
outputvarunits={[unit('m') unit('m')] [unit('m s^(-2)') unit('m s^(-2)')] [unit('m s^(-
2)') unit('m s^(-2)')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
```

```
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
eomtustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));
%% IFO

ifo = ssm(plist('built-in', 'IFO', 'dim', 1));

Ai=ifo.amats;
celldisp(Ai);

Bi=ifo.bmats;
celldisp(Bi);

Ci=ifo.cmats;
celldisp(Ci);

Di=ifo.dmats;
celldisp(Di);

Ai = [0];

Bi = [0 0 0 0 0 0 0 0];

Ci = [0;
      0;];

Di = [1 0 1 0 1 0 1 0;
      -0.9999 1 0 1 -0.9999 1 0 1;];


ifoZOH = ss(Ai,Bi,Ci,Di);
[ifoZOH,G] = c2d(ifoZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0]   [0   0]  [0   0]   [0   0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;-0.9999 1] [1 0;0 1] [1 0;-0.9999 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IFOZOH';
sys.description = 'Ifo discretized with ZOH method';

inputnames    = {'EOM' 'DIST_IFO_READOUT' 'DIST_IFO_POSITION' 'SIGNAL_IFO'};
inputdescription = {'--' '--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'x1' 'x12'} {'x1' 'x12'} {'x1' 'x12'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]
[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IFO'};
outputdescription = {'--'};
outputvarnames ={'x1' 'x12'};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
ifoZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));
```

```
ifoFOH = ss(Ai,Bi,Ci,Di);
[ifoFOH,G] = c2d(ifoFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0]    [0    0]   [0    0]    [0    0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;-0.9999 1] [1 0;0 1] [1 0;-0.9999 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IFOFOH';
sys.description = 'Ifo discretized with FOH method';

inputnames    = {'EOM' 'DIST_IFO_READOUT' 'DIST_IFO_POSITION' 'SIGNAL_IFO'};
inputdescription = {'--' '--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'x1' 'x12'} {'x1' 'x12'} {'x1' 'x12'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]
[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IFO'};
outputdescription = {'--'};
outputvarnames ={'x1' 'x12'};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
ifoFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

ifotustin = ss(Ai,Bi,Ci,Di);
[ifotustin,G] = c2d(ifotustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0]    [0    0]   [0    0]    [0    0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;-0.9999 1] [1 0;0 1] [1 0;-0.9999 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IFOtustin';
sys.description = 'Ifo discretized with Tustin method';

inputnames    = {'EOM' 'DIST_IFO_READOUT' 'DIST_IFO_POSITION' 'SIGNAL_IFO'};
inputdescription = {'--' '--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'x1' 'x12'} {'x1' 'x12'} {'x1' 'x12'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]
[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IFO'};
outputdescription = {'--'};
outputvarnames ={'x1' 'x12'};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];
```

```
% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
ifotustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% ST

st = ssm(plist('built-in', 'ST', 'dim', 1));

As=st.amats;
celldisp(As);

Bs=st.bmats;
celldisp(Bs);

Cs=st.cmats;
celldisp(Cs);

Ds=st.dmats;
celldisp(Ds);

As = [0];

Bs = [0 0 0];

Cs = [0];

Ds = [zeros(st.outputsizes(1), st.inputsizes(1)  ) zeros(st.outputsizes(1),
st.inputsizes(2)  ) zeros(st.outputsizes(1), st.inputsizes(3)  );];


stZOH = ss(As,Bs,Cs,Ds);
[stZOH,G] = c2d(stZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0]   [0]   [0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[0;0] [0;0] [0;0]};

sys.timestep = 0.1;

sys.name = 'STZOH';
sys.description = 'ST discretized with ZOH method';

inputnames    = {'EOM' 'DIST_ST' 'SIGNAL_ST'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = 0;
ssvarnames = 0;
ssvarunits= 0;
ssvardescription = 0;

outputnames    = {'ST'};
outputdescription = {'--'};
outputvarnames = {};
outputvarunits= {};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
stZOH = ssm(plist( ...
```

```
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

stFOH = ss(As,Bs,Cs,Ds);
[stFOH,G] = c2d(stFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0   0]};
sys.cmats    = {[0]};
sys.dmats    = {[0 0 0]};

sys.timestep = 0.1;

sys.name = 'STFOH';
sys.description = 'ST discretized with FOH method';

inputnames    = {'EOM' 'DIST_ST' 'SIGNAL_ST'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = 0;
ssvarnames = 0;
ssvarunits= 0;
ssvardescription = 0;

outputnames    = {'ST'};
outputdescription = [];
outputvarnames = [];
outputvarunits= [];
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
stFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

sttustin = ss(As,Bs,Cs,Ds);
[sttustin,G] = c2d(sttustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0   0   0]};
sys.cmats    = {[0]};
sys.dmats    = {[0 0 0]};

sys.timestep = 0.1;

sys.name = 'STtustin';
sys.description = 'ST discretized with Tustin method';

inputnames    = {'EOM' 'DIST_ST' 'SIGNAL_ST'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = 0;
ssvarnames = 0;
ssvarunits= 0;
ssvardescription = 0;

outputnames    = {'ST'};
outputdescription = [];
outputvarnames = [];
```

```
outputvarunits= [];
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
sttustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% IS

is = ssm(plist('built-in', 'IS', 'dim', 1));

Ais=is.amats;
celldisp(Ais);

Bis=is.bmats;
celldisp(Bis);

Cis=is.cmats;
celldisp(Cis);

Dis=is.dmats;
celldisp(Dis);

Ais = [0];

Bis = [0 0 0 0 0 0];

Cis = [0;
       0;];

Dis = [Dis{1,1}.*ones(is.outputsizes(1), is.inputsizes(1)  )
Dis{1,2}.*ones(is.outputsizes(1), is.inputsizes(2)  ) Dis{1,3}.*ones(is.outputsizes(1),
is.inputsizes(3)  );];

isZOH = ss(Ais,Bis,Cis,Dis);
[isZOH,G] = c2d(isZOH,0.1,'zoh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0 0] [0 0] [0 0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;0 1] [1 0;0 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'ISZOH';
sys.description = 'IS discretized with ZOH method';

inputnames    = {'EOM' 'DIST_IS' 'SIGNAL_IS'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IS'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'}};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
```

```
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
isZOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

isFOH = ss(Ais,Bis,Cis,Dis);
[isFOH,G] = c2d(isFOH,0.1,'foh');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0 0] [0 0] [0 0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;0 1] [1 0;0 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'ISFOH';
sys.description = 'IS discretized with FOH method';

inputnames    = {'EOM' 'DIST_IS' 'SIGNAL_IS'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];

outputnames    = {'IS'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'}};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
isFOH = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

istustin = ss(Ais,Bis,Cis,Dis);
[istustin,G] = c2d(istustin,0.1,'tustin');


sys = struct
sys.params = plist;

sys.amats    = {[1]};
sys.bmats    = {[0 0] [0 0] [0 0]};
sys.cmats    = {[0;0]};
sys.dmats    = {[1 0;0 1] [1 0;0 1] [1 0;0 1]};

sys.timestep = 0.1;

sys.name = 'IStustin';
sys.description = 'IS discretized with Tustin method';

inputnames    = {'EOM' 'DIST_IS' 'SIGNAL_IS'};
inputdescription = {'--' '--' '--'};
inputvarnames = {{'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'} {'tm1_x' 'tm2_x'}};
inputvarunits = {[unit('m') unit('m')] [unit('m') unit('m')] [unit('m') unit('m')]};
inputvardescription = [];

ssnames    = {};
ssdescription = {};
ssvarnames = {{}};
ssvarunits={};
ssvardescription = [];
```

```
outputnames    = {'IS'};
outputdescription = {'--'};
outputvarnames ={{'tm1_x' 'tm2_x'}};
outputvarunits={[unit('m') unit('m')]};
outputvardescription = [];

% Build ssmblocks
sys.inputs = ssmblock.makeBlocksWithData(inputnames, inputdescription,...
inputvarnames, inputvarunits, inputvardescription);
sys.outputs = ssmblock.makeBlocksWithData(outputnames,...
outputdescription, outputvarnames, outputvarunits,...
outputvardescription);
sys.states =  ssmblock.makeBlocksWithData(ssnames, ssdescription,...
ssvarnames, ssvarunits, ssvardescription);
% plist constructors
istustin = ssm(plist( ...
 'amats',sys.amats, 'bmats',sys.bmats, 'cmats',sys.cmats,...
'dmats',sys.dmats, ...
 'timestep',sys.timestep, 'name',sys.name, 'inputs',sys.inputs, ...
 'states', sys.states, 'outputs', sys.outputs));

%% assembling


ltpZOH = assemble([feepsZOH eomZOH ifoZOH isZOH delay dfacs capact accnoise])
ltpFOH = assemble([feepsFOH eomFOH ifoFOH isFOH delay dfacs capact accnoise]);
ltpTustin = assemble([feepstustin eomtustin ifotustin istustin delay dfacs capact
accnoise]);




%% SS

ltpoutSSM = bode(ltpSSM, plist('f',f, 'inputs', {'DIST_FEEPS.sc_x'}, 'outputs',
{'EOM.tm1_x'}));
ltpoutSSM = ltpoutSSM.getObjectAtIndex(1);

ltpoutZOH = bode(ltpZOH, plist('f',f, 'inputs', {'DIST_FEEPS.sc_x'}, 'outputs',
{'EOM.tm1_x'}));
ltpoutZOH = ltpoutZOH.getObjectAtIndex(1);

ltpoutFOH = bode(ltpFOH, plist('f',f, 'inputs', {'DIST_FEEPS.sc_x'}, 'outputs',
{'EOM.tm1_x'}));
ltpoutFOH = ltpoutFOH.getObjectAtIndex(1);

ltpoutTustin = bode(ltpTustin, plist('f',f, 'inputs', {'DIST_FEEPS.sc_x'}, 'outputs',
{'EOM.tm1_x'}));
ltpoutTustin = ltpoutTustin.getObjectAtIndex(1);

%% figure


figure
semilogx(ltpoutZOH.x, abs(ltpoutZOH.y), 'r')
hold on
semilogx(ltpoutSSM.x, abs(ltpoutSSM.y), 'b')
legend('LTP ZOH''LTP SSM')


figure
semilogx(ltpoutFOH.x, abs(ltpoutFOH.y), 'r')
hold on
semilogx(ltpoutSSM.x, abs(ltpoutSSM.y), 'b')
legend('LTP FOH''LTP SSM')


figure
semilogx(ltpoutTustin.x, abs(ltpoutTustin.y), 'r')
hold on
semilogx(ltpoutSSM.x, abs(ltpoutSSM.y), 'b')
legend('LTP Tustin''LTP SSM')


figure
loglog(f, abs(abs(ltpoutZOH.y) - abs(ltpoutSSM.y)) , 'r')
legend('diff')

figure
loglog(f, abs(abs(ltpoutFOH.y) - abs(ltpoutSSM.y)) , 'r')
legend('diff')

figure
loglog(f, abs(abs(ltpoutTustin.y) - abs(ltpoutSSM.y)) , 'r')
legend('diff')

%% ERROR
```

```
nsamples = 10000;

AbsZ = 1/nsamples * sqrt( sum ( (abs(ltpoutZOH.y) - abs(ltpoutSSM.y)).^2  )  )

RelZ = 1/nsamples * sqrt( sum (abs(ltpoutZOH.y) -
abs(ltpoutSSM.y)).^2/sum(abs(ltpoutSSM.y)).^2)


AbsF = 1/nsamples * sqrt( sum ( (abs(ltpoutFOH.y) - abs(ltpoutSSM.y)).^2  )  )

RelF = 1/nsamples * sqrt( sum (abs(ltpoutFOH.y) -
abs(ltpoutSSM.y)).^2/sum(abs(ltpoutSSM.y)).^2)


AbsT = 1/nsamples * sqrt( sum ( (abs(ltpoutTustin.y) - abs(ltpoutSSM.y)).^2  )  )

RelT = 1/nsamples * sqrt( sum (abs(ltpoutTustin.y) -
abs(ltpoutSSM.y)).^2/sum(abs(ltpoutSSM.y)).^2)
```

Here is an example of the building process of the DFACS, FEEPS and IFO sub-models:

DFACS

```
>> dfacs = ssm(plist('built-in', 'DFACS', 'dim', 1))
M: looking for models in /Users/gonzalomorenopousa/Documents/TFC LISA/MATLAB/ltpda_toolbox_2.4/ltpda/classes/+utils/@models/../../../m/built_in_models
M:   running buildParamPlist
M:   running buildParamPlist
M:     constructing from plist
M: looking for models in /Users/gonzalomorenopousa/Documents/TFC LISA/MATLAB/ltpda_toolbox_2.4/ltpda/classes/+utils/@models/../../../m/built_in_models
M:     constructing from plist
M:   running update_struct
M:   running validate
M:   running validate
M:   warning, input named DELAY_ST has all matrices empty, should be deleted
M:   running display
------ ssm/1 -------
      amats: { [10 x10 ] }  [1x1]
      bmats: {   []      []    [10 x2 ]  [10 x4 ]    []      []    }  [1x6]
      cmats: {  [3 x10 ]
                []    }  [2x1]
      dmats: {   []      []     [3 x2 ]   [3 x4 ]   [3 x2 ]   [3 x3 ]
                []      []     [2 x2 ]   [2 x4 ]    []      []    }    [2x6]
   timestep: 0
     inputs:  [1x6 ssmblock]
        1 : DELAY_ST |
        2 : DELAY_IS | tm1_x [m], tm2_x [m]
        3 : DELAY_IFO | x1 [m], x12 [m]
        4 : GUIDANCE | is_tm1_x [m], is_tm2_x [m], ifo_x1 [m], ifo_x12 [m]
        5 : ACTUATION | tm1_x [m s^(-2)], tm2_x [m s^(-2)]
        6 : TESTSIGNAL | sc_x [kg m s^(-2)], tm1_x [kg m s^(-2)], tm2_x [kg m s^(-2)]
     states:  [1x1 ssmblock]
        1 : DFACS | variable_19 [], variable_20 [], variable_21 [], variable_22 [], variable_23 [], variable_24 [], variable_51 [], variable_52 [], variable_53 [], variable_54 []
    outputs:  [1x2 ssmblock]
        1 : DFACS | sc_x [kg m s^(-2)], tm1_x [kg m s^(-2)], tm2_x [kg m s^(-2)]
        2 : DFACS_POSERROR | tm1_x [m], tm2_x [m]
  numparams: (empty-plist)
     params: (empty-plist)
    Ninputs: 6
 inputsizes: [0 2 2 4 2 3]
   Noutputs: 2
outputsizes: [3 2]
    Nstates: 1
 statesizes: 10
 Nnumparams: 0
    Nparams: 0
isnumerical: true
       hist: ssm.hist
   procinfo: []
   plotinfo: []
       name: DFACS
description: DFACS Science Mode 1, All Optical Readouts
    mdlfile:
       UUID: 4a0aa161-ea8e-4486-b26f-f902afb63817
---------------------
```

FEEPS

```
>>feeps = ssm(plist('built-in', 'FEEPS', 'dim', 1))

M: looking for models in /Users/gonzalomorenopousa/Documents/TFC LISA/MATLAB/ltpda_toolbox_2.4/ltpda/classes/+utils/@models/../../../m/built_in_models
M:   running buildParamPlist
```

```
M:   running buildParamPlist
M:   running update_struct
M:   running validate
M:   running validate
>> feeps
M:   running display
------ ssm/1 -------
      amats: { [-14.978580629699529] }  [1x1]
      bmats: {   [1 x3 ]   [0] }  [1x2]
      cmats: { [1]
                [0] }  [2x1]
      dmats: {   []    [1]
                []    [1] }  [2x2]
      timestep: 0
    inputs:  [1x2 ssmblock]
        1 : DFACS | sc_x [kg m s^(-2)], tm1_x [kg m s^(-2)], tm2_x [kg m s^(-2)]
        2 : DIST_FEEPS | sc_x [kg m s^(-2)]
    states:  [1x1 ssmblock]
        1 : FEEPS | x [kg m s^(-1)]
   outputs:  [1x2 ssmblock]
        1 : FEEPS | x [kg m s^(-2)]
        2 : FEEPSNOISE | x [kg m s^(-2)]
  numparams:  (FEEPS_TAU_X=0.066762000000000002, FEEPS_TAU_Y=0.066762000000000002, FEEPS_TAU_Z=0.066762000000000002, FEEPS_TAU_THETA=0.066762000000000002, FEEPS_TAU_ETA=0.066762000000000002, FEEPS_TAU_PHI=0.066762000000000002, FEEPS_XX=1, FEEPS_XY=0, FEEPS_XZ=0, FEEPS_XTHETA=0, FEEPS_XETA=0, FEEPS_XPHI=0, FEEPS_YX=0, FEEPS_YY=1, FEEPS_YZ=0, FEEPS_YTHETA=0, FEEPS_YETA=0, FEEPS_YPHI=0, FEEPS_ZX=0, FEEPS_ZY=0, FEEPS_ZZ=1, FEEPS_ZTHETA=0, FEEPS_ZETA=0, FEEPS_ZPHI=0, FEEPS_THETAX=0, FEEPS_THETAY=0, FEEPS_THETAZ=0, FEEPS_THETATHETA=1, FEEPS_THETAETA=0, FEEPS_THETAPHI=0, FEEPS_ETAX=0, FEEPS_ETAY=0, FEEPS_ETAZ=0, FEEPS_ETATHETA=0, FEEPS_ETAETA=1, FEEPS_ETAPHI=0, FEEPS_PHIX=0, FEEPS_PHIY=0, FEEPS_PHIZ=0, FEEPS_PHITHETA=0, FEEPS_PHIETA=0, FEEPS_PHIPHI=1)
     params: (empty-plist)
    Ninputs: 2
 inputsizes: [3 1]
   Noutputs: 2
outputsizes: [1 1]
    Nstates: 1
 statesizes: 1
 Nnumparams: 42
    Nparams: 0
isnumerical: true
       hist: ssm.hist
   procinfo: []
   plotinfo: []
       name: FEEPS
description: microthrusters linear model using 1st order low pass. No invidual actuator modelling
    mdlfile:
       UUID: a978fedd-fca3-4a57-8b41-044129b92832
```

IFO

```
>> ifo = ssm(plist('built-in', 'IFO', 'dim', 1))

M:   running display
------ ssm/1 -------
      amats: {}  [0x0]
      bmats: {}  [0x4]
      cmats: { }  [1x0]
      dmats: {  [2 x2 ]   [2 x2 ]   [2 x2 ]   [2 x2 ]  }  [1x4]
   timestep: 0.10000000000000001
     inputs:  [1x4 ssmblock]
        1 : EOM | tm1_x [m], tm2_x [m]
        2 : DIST_IFO_READOUT | x1 [m], x12 [m]
        3 : DIST_IFO_POSITION | x1 [m], x12 [m]
        4 : SIGNAL_IFO | x1 [m], x12 [m]
     states:  [1x0 ssmblock]
    outputs:  [1x1 ssmblock]
        1 : IFO | x1 [m], x12 [m]
  numparams:  (IFO_X1X1=0, IFO_X1ETA1=0, IFO_X1PHI1=0, IFO_X1X12=0, IFO_X1ETA12=0, IFO_X1PHI12=0, IFO_ETA1X1=0, IFO_ETA1ETA1=0, IFO_ETA1PHI1=0, IFO_ETA1X12=0, IFO_ETA1ETA12=0, IFO_ETA1PHI12=0, IFO_PHI1X1=0, IFO_PHI1ETA1=0, IFO_PHI1PHI1=0, IFO_PHI1X12=0, IFO_PHI1ETA12=0, IFO_PHI1PHI12=0,
```

IFO_X12X1=0.0001, IFO_X12ETA1=0, IFO_X12PHI1=0, IFO_X12X12=0, IFO_X12ETA12=0, IFO_X12PHI12=0, IFO_ETA12X1=0, IFO_ETA12ETA1=0, IFO_ETA12PHI1=0, IFO_ETA12X12=0, IFO_ETA12ETA12=0, IFO_ETA12PHI12=0, IFO_PHI12X1=0, IFO_PHI12ETA1=0, IFO_PHI12PHI1=0, IFO_PHI12X12=0, IFO_PHI12ETA12=0, IFO_PHI12PHI12=0)
    params: (empty-plist)
   Ninputs: 4
 inputsizes: [2 2 2 2]
  Noutputs: 1
outputsizes: 2
   Nstates: 0
 statesizes: zeros(1,0)
 Nnumparams: 36
   Nparams: 0
isnumerical: true
      hist: ssm.hist
  procinfo: []
  plotinfo: []
      name: IFO
description: Interferometer readout with linear cross-couplings coefficents
   mdlfile:
      UUID: 5cc77d3a-bce1-4625-81fd-21dedfd5157b
