

LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY  
- LIGO -  
CALIFORNIA INSTITUTE OF TECHNOLOGY  
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

<b>Document Type</b> LIGO--T000094-01 - E Sep. 2000
<b>Han2k</b> <b>- End User's Guide -</b>
Hiro Yamamoto and Matt Evans

*Distribution of this draft:*

xyz

This is an internal working note  
of the LIGO Project..

**LIGO Hanford Observatory**  
**P.O.Box 1970; Mail Stop S9-02**  
**Richland, WA 99352**  
Phone (509) 372-2325  
Fax (509) 372-2178  
E-mail: info@ligo.caltech.edu

**California Institute of Technology**  
**LIGO Project - MS 51-33**  
**Pasadena CA 91125**  
Phone (626) 395-2129  
Fax (626) 304-9834  
E-mail: info@ligo.caltech.edu

**LIGO Livingston Observatory**  
**19100 LIGO Lane**  
**Livingston, LA 70754**  
Phone (225) 686-3100  
Fax (225) 686-7189  
E-mail: info@ligo.mit.edu

**Massachusetts Institute of Technology**  
**LIGO Project - MS 20B-145**  
**Cambridge, MA 01239**  
Phone (617) 253-4824  
Fax (617) 253-7014  
E-mail: info@ligo.mit.edu

WWW: <http://www.ligo.caltech.edu/>

# 1 ABSTRACT

Han2k is a program developed to study the LIGO Hanford 2k interferometer, using the LIGO End to End simulation package. This manual explains what Han2k is, what kind of configurations can be simulated, how to run the simulation, how to analyze the result, and some examples showing how to customize Han2k for various different setups. This document is intended to explain how to use the program as it is, and is not intended to explain how to go into the Han2k code and modify it. This document describes the Han2k program released on September 7, 2000, and you need “modeler” (an executable of e2e package) v 1.3 or newer to run this program.

# 2 KEYWORDS

Lock Acquisition, Hanford 2k, COC, LSC, e2e, time domain simulation

# 3 WHAT IS HAN2K

The LIGO End to End simulation package (called e2e hereafter) has been developed to simulate the LIGO detector in the time domain. e2e is a generic calculation environment, like matlab, which has many useful tools built in, including the time domain modal model for the field evolution calculation. Because of its modular design, it can simulate wide ranges of configurations.

The Han2k program has been developed in order to study the lock acquisition design based on e2e. The physics contents in this version of Han2k are as follows:

- The scalar field approximation of electric field
- 6 suspended core optics, no folding mirrors and no mirror thickness
- Parametrized noisy laser inputs to COC
- Parametrized PSL/IOO to simulate the frequency control by the error signal of COC
- BSC stack simulated by a z to z transfer function
- Parametrized seismic motion - local correlations among the stacks in the corner station is approximated by using a common motion caused by a low frequency component of the seismic motion.
- COC LSC servo based on the design of M.Evans - same source code as CDS has implemented at the site.

This version is designed focusing on the study of the length control, and the scalar field approximation is used in calculating the field evolution and the interaction with optics. A study of the mirror misalignment effect has started based on the modified version of Han2k, but it is outside of scope of this document.

Han2k is a collection of files with “.box” file name extension (called box files hereafter). These are like source codes of matlab. The LIGO configuration is “programmed” in these box files. In this document, Han2k is treated as a black box. If you want to customize it, refer to other documents.

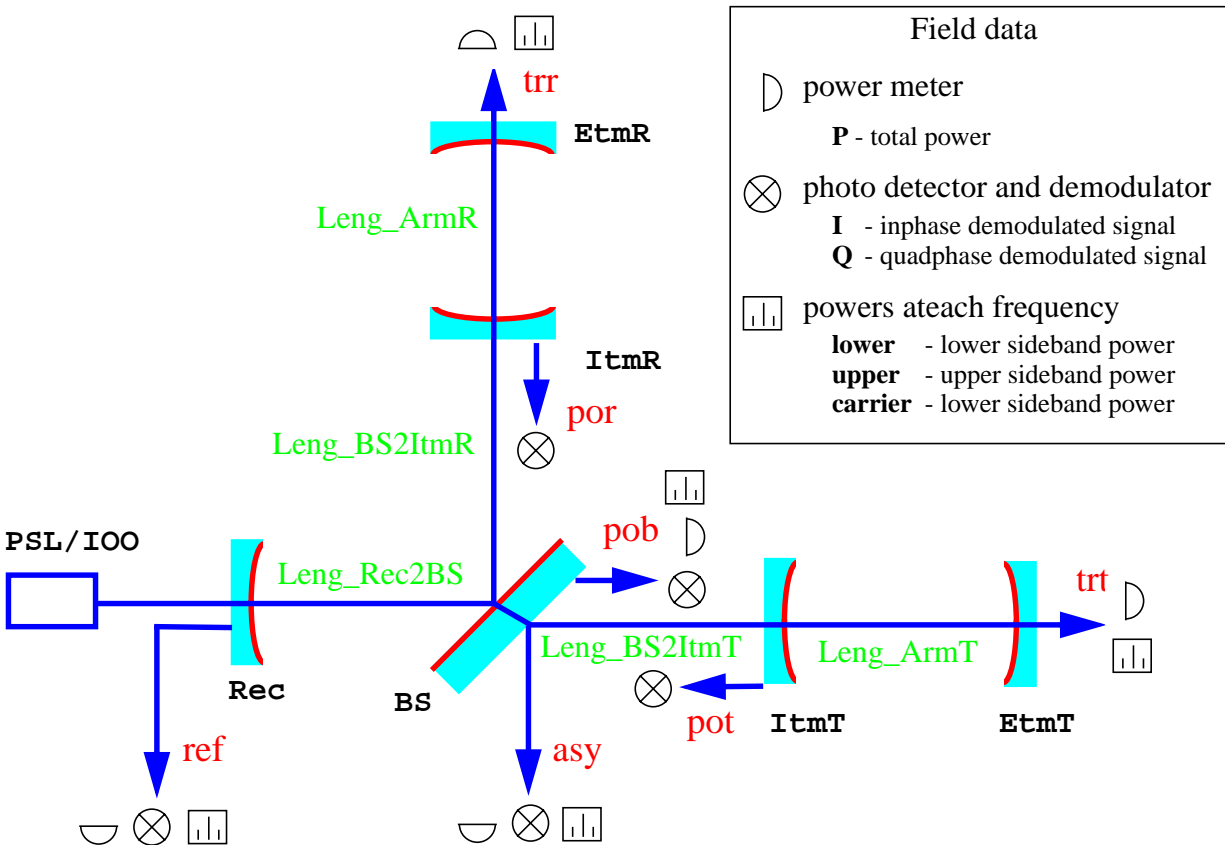


Figure 1: Han2k optics setup

## 4 CONFIGURATION

### 4.1. setup

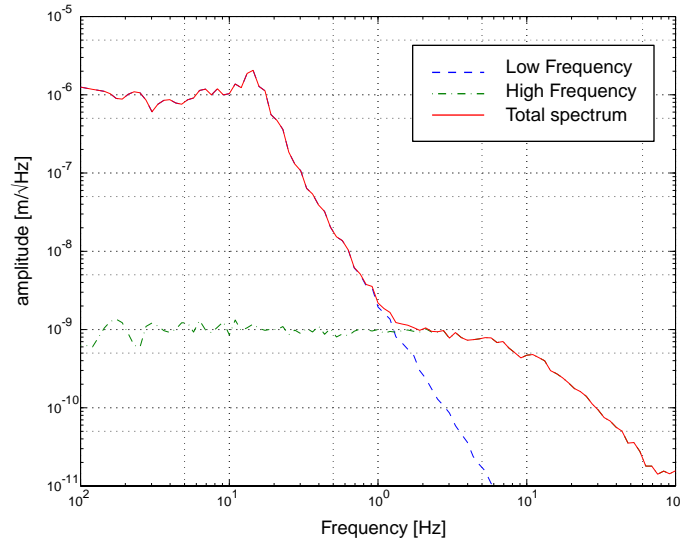
The setup of the optics in Han2k is shown in Fig. 1. Time series data for 7 fields appear in the output of the simulation, **ref** (reflected), **asy** (anti-symmetric), **pob** (pick-off beam-splitter), **pot** (pick off in transmitted direction), **por** (pick off in reflected direction), **trt** (transmitted light in transmitted direction), **trr** (transmitted light in reflected direction).

The symbols in Fig. 1 at a field show what kind of data is recorded for that field. The “power meter” symbol indicated that the total power of that field is recorded and the “demodulator” symbol indicates that the in-phase and quad-phase demodulated signals are recorded. The “power at each frequency” symbol means that the power in the upper and lower sidebands and the carrier are recorded separately, as if by an optical spectrum analyzer. These data are referred to by combining the field name and the type of the data. E.g., **Prt** means the power of the transmitted light in the transmitted direction, and **Qasy** means the quad phase demodulated signal in the asymmetric output port.

The 6 masses, **Rec** (recycling mirror), **BS** (beam splitter), **ItmT** (input test mass in transmitted direction), **ItmR** (input test mass in reflected direction), **EtmT** (end test mass in transmitted direction), **EtmR** (end test mass in reflected direction), are *single suspended mirrors* (simulated by

a simple double pole transfer function) placed on *stacks* (using BSC z to z transfer function) located on *moving platforms* (see next paragraph). Each mechanical system, ground motion, a stack, suspension and an optic, is referred using a name with “Sus” prefixed to the name of the mass. E.g., SusRec is the mechanical system associated with the recycling mirror.

The motion of each *moving platform* (a.k.a ground motion) is simulated in the following way. The horizontal motion spectrum measured at the Hanford site is split into two components, low and high frequency region spectrums. The low frequency component is the measured spectrum up to 1 Hz and drops smoothly in the region above 1 Hz (dashed curve in Fig. 2). The high frequency component is the measured spectrum above 1 Hz, and it is flat below 1 Hz (dot dashed curve in Fig. 2).



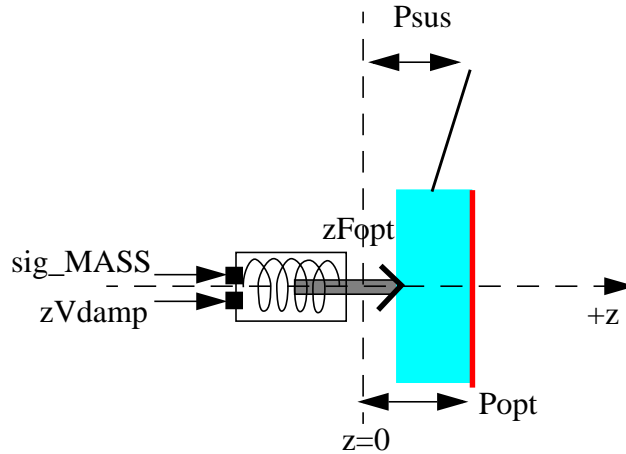
**Figure 2: Seismic motion PSD**

The four *stacks* in the corner station are driven by high frequency spectrum, while the two end mass *stacks* are driven by the sum of low and high frequency components.

$$\begin{aligned}
 \text{SusRec.zPsus} &= \text{SeisHigh\_Rec.SeisScale} * \text{white\_noise} * \text{PSD}(\text{high}) \\
 \text{SusBS.zPsus} &= \text{SeisHigh\_BS.SeisScale} * \text{white\_noise} * \text{PSD}(\text{high}) \\
 \text{SusItmT.zPsus} &= \text{SeisHigh\_ItmT.SeisScale} * \text{white\_noise} * \text{PSD}(\text{high}) \\
 \text{SusItmR.zPsus} &= \text{SeisHigh\_ItmR.SeisScale} * \text{white\_noise} * \text{PSD}(\text{high}) \\
 \text{SusEtmT.zPsus} &= \text{SeisHigh\_EtmT.SeisScale} * \text{white\_noise} * \text{PSD}(\text{high}) \\
 &\quad + \text{SeisLow\_EtmT.SeisScale} * \text{white\_noise} * \text{PSD}(\text{low}) \\
 \text{SusEtmR.zPsus} &= \text{SeisHigh\_EtmR.SeisScale} * \text{white\_noise} * \text{PSD}(\text{high}) \\
 &\quad + \text{SeisLow\_EtmR.SeisScale} * \text{white\_noise} * \text{PSD}(\text{low}) \quad (1)
 \end{aligned}$$

zPsus is the suspension point (see below for the definition), \*.SeisScale are constant parameters, white\_noises are random numbers generated for each occurrence, and PSD(high) and PSD(low) are the high and low frequency components of the power spectrum density of the seismic motion. This is done to simulate the local correlations among the motions of the *stacks* in the corner station. In order to make sure that the low frequency component is not underestimated, the low frequency components used at the end test mass should be enhanced by  $\sqrt{2}$ .

The schematics of a suspended mass system is shown in Fig. 3.



**Figure 3: Suspended Mass**

The +z axis is pointing outward from the coated surface of the mirror. **Psus** is the coordinate of the suspension point, while **Popt** is the coordinate of the coated mirror surface. **zFopt** is the force acting on the mirror which is calculated from the voltage applied on the coil driver, where the saturation effect is included. The voltage applied is the sum of the one from the local damper (**zVdamp**) and the control signal (**sig\_MASS**). The time series of these values are stored in the output data file.

#### MACRO LENGTH :

One important thing about the definition of a distance is to be clarified. Each suspended mass system has its own local coordinate system. All the distances, like **Psus** or **Popt**, are measured with respect to their own local coordinate system. The distances between various local coordinate systems are shown in Fig. 1. E.g., **Leng\_ArmT** is the distance between the inline input test mass and the end test mass. In the simulation calculation, *all these macroscopic lengths are rounded to the nearest integer multiple of the wavelength of the laser*. Without this convention, one has to specify the length with very high accuracy, and that number depends on the specific numerical values of various constants used in the calculation, like the speed of the light. With this convention, a FP cavity, like inline or offline arms, is resonant when the coordinate of the two mirrors, **Popt** of each mirror, are 0, or more strictly speaking when

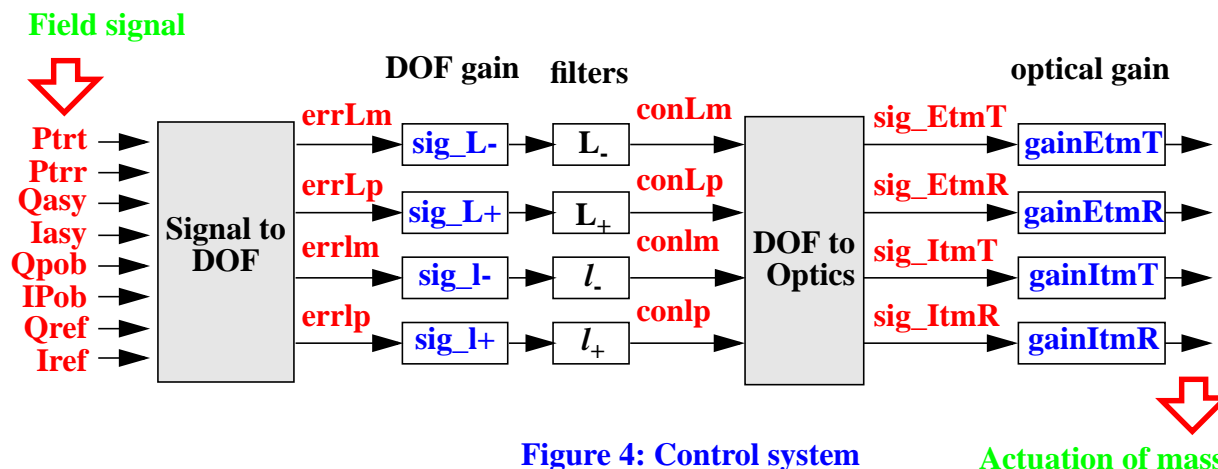
$$Popt(ItmT) + Popt(EtmT) = N \times \lambda(laser)/2 \quad (2)$$

where **N** is an integer. Or, if you want to force the inline arm to be exactly off resonant for a carrier frequency, you set

$$Popt(ItmT) + Popt(EtmT) = \lambda(laser)/4 \quad (3)$$

Because of the definition of the direction of the local coordinate system, *the change of the relative distance is the sum of two local displacements, not the difference*.

The control system is schematically shown in Fig. 4. Six demod signals are converted by “Signal to DOF” to the signals of the 4 DOF, **errLm**, **errLp**, **errlm** and **errlp**. This conversion depends on the state of the IFO, as measured by the transmitted power levels. Filters are applied to the error signals for each DOF, and the resulting control signals, **conLm**, **conLp**, **conlm** and **conlp**, are



converted to control signal for each optic by “DOF to Optics”. All signals shown in red in this figure are stored in the output file and the gain values shown in blue can be easily changed as is explained in Sec. 4.3.

Some distribution packages use dof instead of err for the names of the output of “Signal to DOF” box.

## 4.2. Static parameters

The static parameters of the LIGO detector are stored in “*e2eDB.mcr*”. The file included in this Han2k distribution is for Hanford 2k IFO. The format of a definition line is

name = value [ unit ] “comment”

*unit* and *comment* are optional. A few examples are:

```
ResSBFreq = 29.50588e6 [Hz]  "Resonant SB frequency
ResSBGamma = 0.45    [ ]    "resonant SB modulation depth
ResSBOrder = 2      [ ]    "resonant SB harmonics
```

*Value* can be an expression using numerical literals or previously defined names. The following lines are the definitions of arm lengths.

```
Leng_ArmR = 2009.11 [m] "offline arm length"
Leng_ArmR = Leng_ArmR + 0.1 [m] "Offline arm length"
Leng_ArmT = Leng_ArmR [m] "Inline arm length"
```

The first line is an assignment of the nominal length to the offline arm length. The second line is to change the length by 10 cm, and the third line is to set the inline arm length to the same as the offline arm length.

As is shown in the above example, when the same name is defined multiple times, the later definition is used. If you want to try different values, it is best to add those new definitions at the bottom of the file, in stead of changing the original definitions. Lines which start with “%” are neglected, so you can add comments. Those lines which are sandwiched between lines with “<” and “>” are printed in the window when you run the program. If you insert the following lines at the end of the file, second and third line are printed in the console window to remind the user that this is a special run.

<

```
% Simulating of the one arm run by making the end test mass almost transparent
L_EtmR = 1-T_EtmR [] "mimic closed gate valve or misaligned optic
>
```

Strictly speaking, e2eDB.mcr is a file where macro names are defined. Macro names defined in this file can be used anywhere numerical values are used. If there are any values you prefer to use symbolic names in the set up of the dynamics explained in Sec. 4.3., that can be defined here. Some constants like PI are defined automatically. You can find the current macro definitions by typing

```
e2emacro.
```

### 4.3. Defining dynamic set up

You can set up the dynamic conditions and the setup of the control systems of the simulation by modifying a file named “*Han2k.par*”. The format of a definition line is

```
box1.box2...boxN.paramName = value
```

and comment lines begin with “%”. boxN is the name of a system which directly encloses the target “paramName”, box(N-1) encloses boxN, .., and box1 encloses box2. A few examples will clarify.

```
gainEtmT = 1
sig_L-.gain = 1
SusEtmT.Damper.gain = 1
SusItmT.Damper.gain = 1
```

The first and second define the gains of the controls shown in Fig. 4, and the third and fourth define the gains of the local Damper of the suspended mass system of the input and end test masses. If you want to set the gain of all the local dampers to the same value, you do so in the following way:

```
Dampler.gain = value
```

With value = 0, all local dampers are disabled. As you can see from this example, the value assignment is applied to all parameters whose names end with the name specification, box1...boxN.paramName. If the following line is included, all settings with the name “gain” are set to zero.

```
gain = 0
```

This could result in an unpredictable result.

#### **FIDLDE :**

A function Fiddle is implemented in Han2k in order to define a common time dependence in various parts of the simulation. The definition is as follows:

$$\begin{aligned} \text{Fiddle}(x) = & \textit{gain} * x \\ & + \textit{offset} + \textit{speed} * \textit{time} \\ & + \textit{amp} * \sin(2 * \pi * \textit{freq} * \textit{time} + \textit{phase}) \\ & + \textit{noiseAmp} * (\text{white noise with lowpass filter at 1Hz}) \end{aligned} \quad (4)$$

One can specify those seven parameters shown in italic in this equation, and “time” is the time elapsed since the start of the simulation. The default values of these parameters are all

zero except for the parameter “*gain*” whose default value is 1. By assigning proper values to these parameters, linear motion, sinusoidal motion and random noise can be simulated.

Sweep.z is a Fiddle type with the input value x set 0.

```
Sweep.z.offset = -2e-8
Sweep.z.speed = 2e-6
```

With these settings for z, the output is a linear motion with the initial location -2e-8.

```
Sweep.z = -2e-8 + 2e-6 * time
```

The mirror positions, like SusEtmT.zPopt, is a summation of this Sweep.z and the coordinate dynamically determined by the suspension point motion and the control force on the mirror. The default setting is to simulate a suspended mass with control force acting on it. If you set SeisScale for the mirror (SeisHigh\_EtmT.SeisScale) to 0 to nullify the suspension point motion and set the optical gain (e.g., gainEtmT) and the Damper gain (Damper.gain) to 0 to nullify the actuation of the mass, then the mirror motion due to the dynamics does not contribute at all, and you can define the motion of the mirror using the Fiddle parameters as is explained above.

In the Han2k.par file included in the distribution package contains the follow lines.

```
SeisHigh_BS.SeisScale = 0
SeisHigh_Rec.SeisScale = 0
SusBS.Damper.gain = 0
SusRec.Damplere.gain = 0
```

There is no control force acting on BS and Rec mirrors, so these masses do not move.

```
% SusRec.Sweep.z.offset = 2.66e-7
% SusRec.Sweep.z.speed = -0.36e-6
% SusBS.Sweep.z.offset = -1.746e-8
% SusSB.Sweep.z.speed = 1.61e-6
```

All these lines start with the comment symbol “%”, so they are neglected. If you remove the comment symbol from the first line, Recycling mirror is forced to locate at 0.266  $\mu$  inside from the nominal position. If you also active the second line, the recycling mirror now moves at a constant speed. When the 3rd and 4th lines are activated, the beam splitter starts to move at a constant speed.

DOF gains, like sig\_L-, are Fiddle types. With the default values and assigning a value to the gain of the Fiddle, it behaves as a multiplier. But if you assign a finite value to noiseAmp, you can introduce an electronic noise.

## 5 HOW TO RUN

### 5.1. modeler, the time domain simulation program

“*modeler*” is the program which does the time domain simulation. The main input to modeler is a collection of box files in which the configuration of the system to be simulated are defined. modeler also reads in e2eDB.mcr and .par file to get the all necessary values for the simulation. When you run modeler, it requests you several inputs, including the name of the main box file and



the par file for it. Those queries and replies are stored in “*Han2k.in*”, and you don’t need to type inputs yourself. Instead, you just type the following and the program starts running.

```
cat Han2k.in - | modeler
```

In a few minutes, modeler asks if you want to continue, and type “no”. Then the execution is completed, and you will find three new files created, *Han2k.dat*, *Han2k.dhr* and *Han2k.set*.

*Han2k.dat* contains all the output data, the first column is the time, and all others are the time series of various data. The format is as follows:

```
time-1  dat1-1  dat2-1  dat3-1
time-2  dat1-2  dat2-2  dat3-2
time-3  dat1-3  dat2-3  dat3-3
....
time-N  dat1-N  dat2-N  dat3-N
```

time-I is the time and datJ-I is the value of datJ in the Ith set of data. *Han2k.dhr* is a list of these data in the order from left to right. The *Han2k.dhr* will be as follows:

```
time
Han2k.gainOnOff
Han2k.sig_ItmR
...
Hak2k.2kDetector.IFO_SUS.SusItmR.zPopt
Hak2k.2kDetector.IFO_SUS.SusItmR.zFopt
Hak2k.2kDetector.IFO_SUS.SusItmR.zPsus
Hak2k.2kDetector.IFO_SUS.SusItmR.Damper.zVdamp
```

From this, you find that the first column in *Han2k.dat* is time, second is a variable called gainOnOff (this is 0 when the LSC servo is off, and 1 when on), third is the control signal on the ItmR mass, 4th from the end is the mirror position of ItmR, 3rd the force acting on it, 2nd the suspension point and the last is the local damper voltage.

There are bunch of outputs grouped under the name of SigMon, like *Han2k.SigMon.Pasy*, *Iasy* and *Qasy*. These are the power, inphase and quad phase demodulated error signals in the asymmetric output port. Another group of interest is PowerSplit\_, like *Han2k.2kDetector.PowerSplit\_pob.lower*, *upper* and *carrier*. These are the power of the lower and upper sidebands and the carrier measured at the pickoff of the beam splitter.

*Han2k.set* is a file which contains all the setup information describing how the program run. It includes the date and time when the program run, the initial random number seed, all the macro setting, in addition to all the parameters used in all the box files. Simulations with identical set files should produce identical results.

## 5.2. Tweaking the run

### 5.2.1. run longer

When you want to run the simulation longer, there are two ways. In the *Han2k.in* file, there is a line you answer how long you want to simulate.

```
% Simulation time (s) (def=1,[0:INF]) >>
```

0.1

First line is a prompt line, which is included as a comment line so that you can find what you are answering, and the second line if you answer to that question. You can use a text editor to change your answer to a larger value, like 5. The unit of time is seconds, so you change the simulation time from 0.1 second to 5 seconds.

Another way to extend the run is to answer “yes” when you are asked if you want to “continue”. Then you are asked how many more seconds you want to simulate, and how often do you want to save the data. If you answer 100 to the second question, every 100th events are stored in the file. When you use this method to extend the run, you can modify the Han2k.par file before you answer “yes”. For example, you run the system without any noise for 1 second to let the cavity being filled under the perfect locked condition, and you can continue the run with all the noises turned on to see how the system is fragile or robust against the disturbance by the noise. Or you can run the system with only part of the control signals active, and later you can manually turn on the rest of the controls.

### 5.2.2. store data in a binary file

When you run modeler as is explained above, the data is stored in an ascii file. When the amount of the output data become large, it takes long time to read the data for the analysis. You can store output data in a binary file by using `-bin` run time option.

```
cat Han2k.in - | modeler -bin
```

If you want to convert the binary data file into an ascii file, type

```
e2ebinLoader datafile.dat
```

Then two files, datafile.asc and datafile.dhr, are created. datafile.asc is the ascii file which contains the data, and datafile.dhr contains the names of the data in the order they appear in the datafile.asc.

### 5.2.3. change the order of the data in the output file

When you start analyzing data, you may find that the default output order is not convenient. E.g., you may want all the error signals and power located at the beginning of the output file. You can change the order of the output by changing the order of the names in Han2k.dhr file. E.g., if you edit Han2k.dhr file and move all the 15 Han2k.SigMon names to just below the line of “time”, then modeler stores the data in the order specified by Han2k.dhr. More precisely speaking, modeler looks for a file xxx.dhr when it creates a file named xxx.dat. If there is one, then modeler reads in the list of names from xxx.dhr and tries to respect that order as much as possible. If there is no file of that name, modeler uses its internal rule to determine the order of data.

### 5.2.4. Define a seed of the run

If you want to run a program with a given seed, use `-seed` run time option.

```
modeler -seed 123
```

This is a way to force the initial seed of the run to be 123. When there is no initial seed specified, the clock data is used to generate a random initial seed. If you want to find what was the initial seed of a run, you can find it in the .set file.

### 5.2.5. Use additional macro data file

If you want to use additional macro data file, you can specify the file with a runtime option `-db`.

```
modeler -db filename
```

The macro definitions in the file `filename` is loaded after all `e2eDB.mcr` files are loaded, so the macro definitions in this file override all definitions done before. The name of the file can be anything.

## 6 HOW TO ANALYZE DATA USING MATLAB

### 6.1. Basic

When you generated an ascii data file, you can load the data into matlab using load:

```
dat = load('asciidata.dat');
```

If you generated a binary file, you need `e2ebin.m` which is included in Han2k distribution package. You can load the data in the following way:

```
[dat, lgnd] = e2ebin('binarydata.dat');
```

The binary file has the data titles included in the file, and you can extract that as the second argument of the return value of `e2ebin` function. `lgnd(1) = 'time'`, `lgnd(2) = 'Han2k.gainOnOff'`, etc.

Once the data is loaded, you can plot the time series of various data.

```
>> subplot(2,1,1)
>> plot(dat(:,1),dat(:,70)+dat(:,74),dat(:,1),dat(:,54)+dat(:,58),'--')
>> legend('Offline arm length', 'Inline arm length')
>> subplot(2,1,2)
>> semilogy(dat(:,1),dat(:,46),dat(:,1),dat(:,49),'--')
>> legend(char(lgnd(46)),char(lgnd(49)))
```

These matlab commands generate the plots in Fig. 6. The top figure is the time evolution of two arm lengths (solid : offline arm, dashed : inline arm) and the bottom is the time evolution of the carrier power in each arm. In the simulation, the length control is turned on at 1 second (`local_delay` in `Han2k.par` is the time when the lock loop is turned on). The offline arm is locked at around 1.3 second, the inline arm is locked at 2.95 second, and the powers in these arms go up to the full power in 0.1 second.

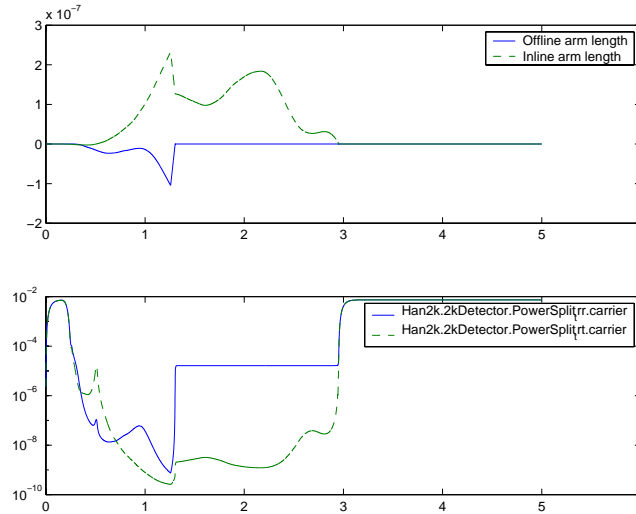
### 6.2. using plotting aids

Han2k comes with several matlab m files to make the analysis easier. In matlab, you can use help to find the details of the usage of those functions.

```
[dat, lgnd] = modelerLoad('Han2k');
[dat, lgnd] = modelerLoadBin('Han2k');
```

`modelerLoad` loads data from an ascii file and titles of data and returns in `dat` and `lgnd`. `modelerLoadBin` does the same for binary data file.

```
initCols
```



**Figure 5: Lock of two arms**

initCols groups related data into combined arrays so that it is easier to view related data. [initCols needs the data stored in a variable named “dat”, and lenend in “lgnd”.] The list of group is available in the help message. A few examples are:

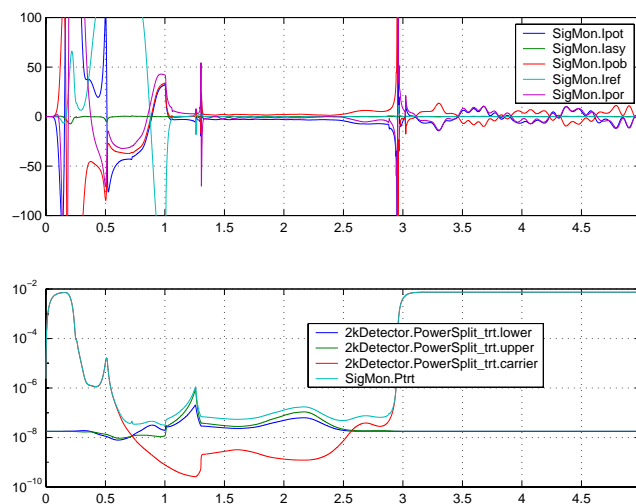
IFO Output Signals

c\_sigI - in-phase demod signals  
 c\_sigQ - quad-phase demod signals  
 c\_sigP - power signals

After this initialization, you can use dualPlot to plot two groups of data in one window. E.g.,

```
dualPlot(dat, lgnd, c_sigI, c_trtPow)
```

will draw the plots show in Fig. 6. The top plot is the time evolution of all inphase demodulated signals and the bottom one is the evolution of the powers of the transmitted power in the inline arm.



**Figure 6: dualPlot**

NO-DRAFT

## 7 EXAMPLES

## 8 LOOKING INTO THE BLACKBOX

to be filled

### Reference

[1] to be filled

LIGO-DRAFT