

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

<b>Document Type</b> <b>LIGO-T970194- - E</b> Nov. 1997
<b>Organization of End-to-End model</b>
Biplab Bhawal, Matt Evans, Edward Maros, Malik Rahman and Hiro Yamamoto

*Distribution of this draft:*

xyz

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

**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

**Massachusetts Institute of Tech-**  
**nology**  
**LIGO Project - MS 20B-145**  
**Cambridge, MA 01239**  
Phone (617) 253-4824  
Fax (617) 253-7014

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

# 1 ABSTRACT

xxx

# 2 KEYWORDS

xxx

# 3 DEFINITION OF WORDS

# 4 OVERVIEW

xxx

# 5 MODULAR NATURE

Systems are built from modules.

# 6 HIERARCHICAL ORGANIZATION

Boxes are used to organize modules.

# 7 IMPLEMENTATION SCHEME

## 7.1. Overview

This section describes how the system to be modeled and its associated data flow network is constructed and executed.

## 7.2. Construction - *find modules, create ports and connect*

The two primary operations that take place during construction of a system are the creation of the system's components and their interconnection. (refer to section which describes Box Description files) Components are created as they are found by the parent box in its "Add\_Submodules" section. Initialization may be done at the time of creation or at some later point through the use of the box "Settings" section title. It can be that explicit initialization may not be called at all, and minimum initialization needs to be done in the constructor of the module class.

The following two descriptions, desc-1 and desc-2, define the same configuration consisted of one digital filter who has one pole at -1 and one zero at -3.

```
desc-1 Add_Submodules { digital_filter df { pole = -1; zero = -3} }
```

```
desc-2 Add_Submodules { digital_filter df }
      Settings df { pole = -1; zero = -3}
```

When `df` is created using the first form, the module is created and its initialization routine, `sub_load` and/or `sub_sub_load`, are called with `{ pole = -1; zero = -3}` as the inputs. When `df` is created using the second form, the initialization routine is called one when the module is created with empty input, and is called again with `{ pole = -1; zero = -3}` as the inputs. The initialization of the module can be called multiple of times with different values of inputs, and the module should be prepared for it.

The interconnection of components takes place when a parent box encounters an “Add\_Connections” section title. [As mentioned earlier,] the connection syntax is

```
[source_component] [source_port] -> [sinc_component] [sinc_port].
```

If the port name is missing, number 0 is assigned to it. If the port name is an integer number, it is synonymous with “input”+number or “output”+number, e.g., “input0” or “output21”. The following two sections mean the same connection.

```
Add_Connections { mod_a -> mod_b 21 }
Add_Connections { mod_a output0 -> mod_b input21 }
```

The first action the parent box takes is to find the two components among its children. Succeeding in this both the source and sinc components are asked to find or create their respective ports ( `module::find_or_add_port(const string &pname)` ). This will fail if either component fails to recognize the port name in the request. If it is successful the process is completed by connecting the ports ( `module::connect(const string& pfname, module* dest, const string& pname)` ).

### 7.3. Data Routing

Once the system is fully constructed and connected, modules can begin to track down their input data sources.

### 7.4. Time Evolution Sequencing

The above data routing system leaves no room for buffering in data flow, thus the sequence in which the modules evolve their internal state and produce outputs is crucial.

### 7.5. Configuration

Now the model is ready to run, but none of the particulars have been specified. One must specify the time step and all of the input values.

#### 7.5.1. Simulation time step, $\Delta t$

The time step  $\Delta t$  is provided by the user. This should be a factor of several smaller than the time step of the physics process of interest. If the simulation is for the pendulum,  $\Delta t$  may be 10ms,

while if the simulation is for the details of the cavity response,  $\Delta t$  may be  $1\mu\text{s}$ . For the given time step, each module should calculate the response.

Some kind of tricks or approximation may be needed to achieve reasonable speed for the simulation. E.g., when the Michelson cavity, with arm lengths  $L_1$  and  $L_2$ , is made using mirror modules (beam splitter and two end mirrors) and two field propagators, the following calculation is done. Assume the case that the time step is chosen to be  $(L_1+L_2)/(2*c)$ , and  $L_1$  and  $L_2$  are slightly different. The field in each arm propagates from the beam splitter to the arm mirror in one time step, and bounces back to the beam splitter in the next time step. The difference of the arm lengths are taken care by the phases gained in each trip, which is based on the exact length at proper time.

This phase calculation is divided into two pieces. The length assigned to the field propagator is assumed to be an integer multiple of the carrier wave length. The difference between the exact length and the rounded length assigned to the field propagator is taken care by the mirror. I.e., microscopic changes of the mirror motion is taken care by the additional phase calculated by the mirror to account for the deviation from the resonance condition.

## 7.6. Execution

The model evolves by evolving each of the modules in the predetermined tick sequence (and preticking the delay modules, see Sec. 7.7.2. for details). Each full run through the pretick and tick queues constitutes a step forward in time. This process may be iterated as many times as desired and the outputs read as frequently as one wishes, thus producing a time series for each of the monitored outputs.

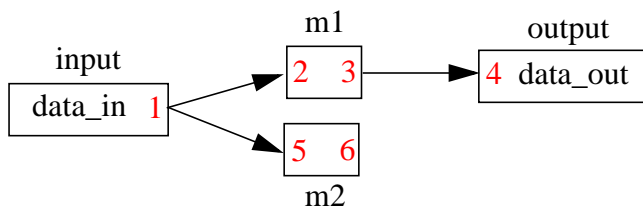
## 7.7. Modules

### 7.7.1. Input and output modules

There are two modules which have only an input or only an output, but not both. The “data\_in” module is used to bring data into the simulation from outside sources, frequently the user. “data\_in”, as viewed from inside the simulation, has a single output, which may be of any type. Similarly, the “data\_out” module is used to get data out of the simulation. In the simulation it behaves as a module with a single input, which may be of any type. Each of these modules serve a special function in the simulation.

A “data\_in” is special in that it can produce its output independent of all other modules in the simulation, since, by definition, it takes no input from the simulation. This feature makes “data\_in” modules convenient points from which to initiate data flow, i.e. “data\_in” modules are data flow sources.

A “data\_out” module, on the other hand, is a data flow sink. All of the data generated by the simulation serves the single purpose of producing the inputs for the “data\_out” modules in the simulation, since it is from these modules that the data are retrieved for analysis by the user. A corollary to this is that any data flow path which does not lead to a “data\_out” need not be, and in



**Figure 1: Input and output**

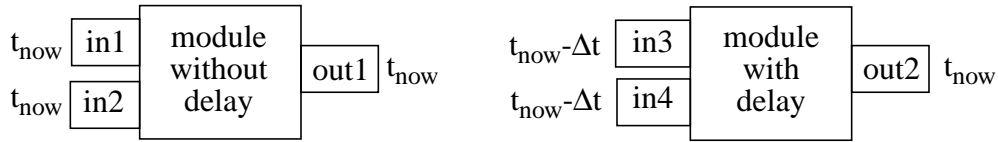
fact is not, simulated. In the example shown in Figure 1, the simulation goes as follows. First, “input” is “ticked”. Ticking a module, by way of the `module::tick()` member function, causes it to perform whatever processing is necessary to update its outputs for this simulation cycle. (In the case of a “data\_in” no processing is necessary.) After “input” is ticked, output #1, and thus input #2, is ready for use by “m1”. “m1” is then ticked, producing output #3. Now input #4, used by “outout” is ready and, since that is the only “data\_out” in the simulation, time evolution for this simulation cycle is complete. In this example, “m2” was not ticked because it is not needed to produce the requested output. If another “data\_out” module is added and is connected to output #6, then “m2” would also be ticked.

### 7.7.2. Delay in modules

As eluded to earlier, the simulation proceeds by cycling through a list of modules and requesting that each one evolves its internal state and its outputs. To facilitate further discussion, the size of the step taken forward in time in one cycle, the “time step”, will be referred to as “ $\Delta t$ ” and the time represented by the current simulation cycle will be called “ $t_{\text{now}}$ ”. As one might expect, the time represented by the previous simulation cycle is “ $t_{\text{now}} - \Delta t$ ” and so on. With this terminology in hand we are ready to discuss delay in modules and its importance in simulation.

There are two kinds of modules, distinguished by how the output depends on the input, delay and non-delay. A non-delay module simulates a process in which output at  $t_{\text{now}}$  depends on the input at  $t_{\text{now}}$ . The mirror module is of this kind, since the reflected and transmitted field at  $t_{\text{now}}$  are calculated using the input fields at  $t_{\text{now}}$ . The digital filter module is also of this kind, since the output at  $t_{\text{now}}$  depends on past ( $t_{\text{now}} - \Delta t$  and earlier) input data as well as input at  $t_{\text{now}}$ .

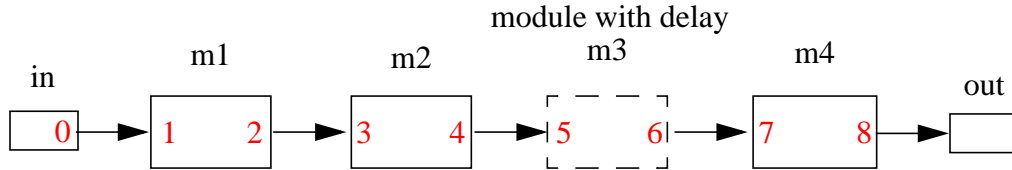
A module with delay simulates a process where the current output does not depend on the current input, but is determined solely by the input at time  $t_{\text{now}} - \Delta t$  and earlier. The “propagator” module, which simulated the propagation of a field over some distance, is of this kind. When  $\Delta t = (L/c) / n$ , where  $L$  is the length of the propagator,  $c$  is the speed of light, and  $n$  is an integer, the output of the prop at  $t_{\text{now}}$  is determined by the input at  $t_{\text{now}} - n\Delta t$ , i.e.  $t_{\text{now}} - (L/c)$ . Another example of a module with delay is the module apply named “delay”, whose sole purpose is to place a delay in a specific place in the simulation. A delay module may be used, for instance, to simulate the delay in a control signal sent over a long distance (i.e. from the corner station to the end test mass). This module also plays an important role in resolving loop ambiguities in the simulation, which will be explained in the next section.



**Figure 2: Input-output Dependence**

The dependence of the output of a module on its inputs is summarized in Figure 2. For those modules without delay, the output at  $t_{\text{now}}$  depends on the input at  $t_{\text{now}}$ , while for those with delay, the output does not depend on the input at  $t_{\text{now}}$ , and the output can be calculated using existing data up to  $t_{\text{now}} - \Delta t$ .

All modules should implement their own member function `action()`, called from `module::tick()`, and all modules with delay should implement their own member function `preaction()`, called from `module::pretick()`, both of which are provided as virtual functions in the base module class. At the beginning of the simulation cycle all modules with delay are “preticked”. At this point the module should calculate their output using existing data up to  $t_{\text{now}} - \Delta t$ , so that after this call, the outputs of these modules are values for  $t_{\text{now}}$ , as shown on the right side of Figure 2. After the pretick comes the tick. When a module is ticked it should calculate update its internal state and, if it hasn’t already, it should update its output for time  $t_{\text{now}}$ . This leads to the situation show on the left side of Figure 2.



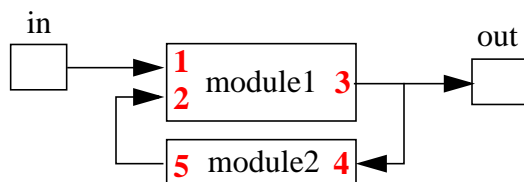
**Figure 3: Simple simulation**

As a simple example, the evolution in Figure 3 goes as follows. In this example, “m3” has delay and all other modules do not. First “m3” is preticked. The value at output #6 is updated using data up to  $t_{\text{now}} - \Delta t$ . Then the other modules are ticked in order. First, “m4” is ticked, and using the value at #7, from #6 which was updated in the pretick, updates output #8. At this point the input to “out” is ready, but the cycle is not over, since input #5 has not yet been calculated. Though this input was not needed to produce the output of this simulation cycle, it may be needed in the next cycle. Input #5 is produced by first ticking “in” which does whatever necessary to update output #0 to  $t_{\text{now}}$ , followed by “m1” and then “m2”. Lastly, “m3” is ticked. This call does not change output #6, because it was updated to  $t_{\text{now}}$  in the pretick phase. Instead, “m3” uses the tick to update its internal state, if necessary, so that it can produce output #6 when pretick is called in simulation cycle  $t_{\text{now}} + \Delta t$ .

### 7.7.3. Loops

In a framework like Adlib, where discrete time steps are used to simulate the time evolution, loops may result in an ambiguity as to where the simulation should start. E.g., in Figure 4, the simulation may start by ticking “module2” to produce output #5 using input #3 from the previous cycle, then using data from “in” and #5, calculate the output #3. Alternatly, it may start by using

data from “in” for input #1 and data from the previous cycle for input #2 to calculate output #3, then use this output to calculate #5 for this cycle. The result in either case is that an implied delay has been added to the loop. The ambiguity is in the location of this delay.



**Figure 4: Loop**

If any module in the loop has delay this ambiguity ceases to exist. Let’s analyze this example for the case that “module1” has delay. When the input-output dependence in Figure 2 is taken into account, the only solution is to start by evaluation output #3, because only output #3 can be evaluated without input at  $t_{\text{now}}$ . All other outputs depend on inputs at  $t_{\text{now}}$ , none of which exist.

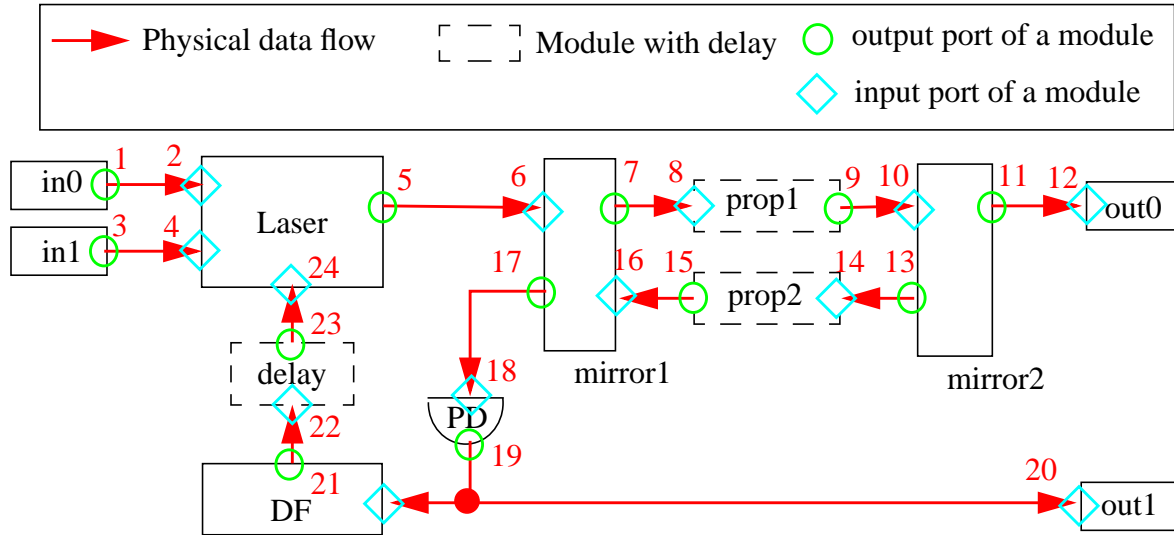
If, on the other hand, no module in a loop has delay, there is no guideline as to where to start. This is an intrinsic ambiguity of the simulation framework, and the output of the simulation may in general depend how this ambiguity is resolved. The dependence should be small when the variables in the loop change by a only small amount in a single simulation cycle, i.e.  $1/\Delta t$  is beyond the bandwidth of the loop.

For a given simulation configuration in which this ambiguity exists, Adlib inserts an implied delay of one time step at some arbitrary location in the loop and prints out a warning message. It is strongly recommended that one places a “delay” module in a most appropriate location in the loop (based on an educated guess, aesthetics or whatever), such that the warning is not given. The delay time may be set to zero (the default), but since a “delay” module always delays a signal by at least  $\Delta t$ , this is sufficient to resolve the ambiguity. While this does not prevent dependence of the output on the location of the delay, it at least makes this dependence testable and make the output reproducible.

LIGO-DRAFT

## 7.8. Example

In this section, an example in Figure 5 is used to explain how the time evolution occurs in the



**Figure 5: Example of execution**

model. The following explanation is done to help to understand the source code as well. All of these constructions and time evolution are done in the application framework class `modeler_base`. There are three important queues for constructing and executing the time evolution loop, **tick** queue, **pretick** queue and **tbd** queue. The tick queue stores a pointer to each computational module in the simulation. Modules appear in the queue in the order of execution, i.e., the tick queue determines the time evolution sequence. The pretick queue stores pointers to modules which simulate processes with time delay. It is called the “pretick” queue because modules in this queue are called upon to produce their outputs at the beginning of each time step. After the outputs of these modules are updated to  $t_{\text{now}}$ , the modules in the tick queue are instructed, in order in the queue, to use their inputs (which should all be at  $t_{\text{now}}$ ) to evolve their internal states and produce outputs at  $t_{\text{now}}$ . The tbd queue is used during the construction of the other two queues, and it keeps modules which are to be “handled”.

After the system has been constructed, the time evolution sequence must be determined. This process begins by placing all output modules in the tbd queue. The process of time evolution sequencing is complete when the tbd queue is empty. The tick queue and pretick queue both start empty. In this example, the initial state of the three queues are as follows;

```
tick = []
pretick = []
tbd = [out0, out1]
```

The construction proceeds recursively. Start by taking the first module from the tbd queue, move backward along the data flow into this module, until you reach a module with no input or a module which has delay. Either of these modules can produce output at  $t_{\text{now}}$  without inputs at  $t_{\text{now}}$ , and thus the simulation sequence can start by using these outputs.

The first module in the tbd queue is out0. Go backward along the data connection arrow, i.e., from #12 (the port number 12 in Figure 5: Example of execution) to #11. The module mirror2 has no delay, so go to the input, #10, and move backward to #9. prop1, the field propagator, is a module with delay. Place prop1 in the pretick queue (because this has delay) and in tdb queue (because its inputs will be examined later). Since prop1 is in the pretick queue, the recursive trace ends here, follow the data flow back to mirror2. There are no more inputs to be examined, so place mirror2 in the tick queue (because this one needs to be executed during time evolution) and continue forward in the data flow to out0. Its inputs have now been fully examined, put it into the tick queue. Now the queues are as follows.

```
tick = [mirror2, out0]
pretick = [prop1]
tbd = [out1, prop1]
```

Start again from the first module in the tbd queue, i.e., out1. Move back to #19. The photo diode has no delay and only one input, #18. Move against the data flow to #17. Mirror2 has no delay so proceed to one of its two inputs, #16 or #6. In this example, pick #16. This leads to #15. prop2 has delay, so stop here. Place prop2 into the pretick queue and tbd queue, then return to mirror1. “mirror1” has one more input, #6, so go from there backward to #5. “Laser” has two inputs. Choose #2, and move to #1. “in0” has no inputs, so stop there and put “in0” into the tick queue and move back to “Laser”. Do the same for “in1”. The last input is connected to “delay”, and place it in the tbd and pretick queues. Now all the inputs of “Laser” have been examined, and the queue status is

```
tick = [mirror2, out0, in0, in1]
pretick = [prop1, prop2, delay]
tbd = [prop1, prop2, delay]
```

Start the return trip by placing “Laser” in the tick queue. Then go forward from output #5 to mirror1. All inputs of mirror1 have been handled, so put mirror1 in the tick queue, and return to “PD”, place it in the tick queue, followed by “out1”.

```
tick = [mirror2, out0, in0, in1, Laser, mirror1, pd, out1]
pretick = [prop1, prop2, delay]
tbd = [prop1, prop2, delay]
```

This data flow path is complete, so remove “prop1” from the tbd queue and examine its inputs. There is one input, #8, and that is connected to mirror1 which has already been analyzed, so place prop1 in the tick queue. Do the same for prop2. Now the queue status is

```
tick = [mirror2, out0, in0, in1, Laser, mirror1, PD, out1, prop1, prop2]
pretick = [prop1, prop2, delay]
tbd = [delay]
```

The last module in the tbd queue is “delay”. Input #22 is connected to DF which receives its input from “PD”. But “PD” is done, so place “DF” in the tick queue, followed by “delay”.

```
tick = [mirror2, out0, in0, in1, Laser, mirror1, pd, out1, prop1, prop2, DF, delay]
pretick = [prop1, prop2, delay]
tbd = []
```

Now the tbd queue is empty and the time evolution sequence, represented by the tick queue, is complete.

Consider an alternative configuration in which there is no “delay” module. When tracing backward from “Laser” input #24, “DF” and then “PD” are encountered. However, “PD” is

already in the process of examining its inputs. This indicates that an ambiguous causal loop exists. When this happens, a warning is printed out and “DF” is placed after “in1” in the tick queue. This is equivalent to placing a one-time-step delay between “DF” and “PD”.

Returning to the configuration in Fig.2, with the tick and pretick queues are constructed, the simulation may proceed. It goes as follows. First, the member function pretick() is called on all modules in the pretick queue. When called, they should produce their outputs for time  $t_{\text{now}}$  using only internal data, which resulted from inputs at time  $t_{\text{now}} - \Delta t$  and earlier. Then the member function tick() is called on each module in the tick queue in order. The input to each module will be at  $t_{\text{now}}$  when the module is called, and the module is responsible for producing output at  $t_{\text{now}}$ .

In this example, *mirror2* is called first to calculate the output using #10. *prop1* is in the pretick queue, so output #9 has already been calculated. The input to *out0* is now at  $t_{\text{now}}$  and may be used as needed. *in0* and *in1* prepare data for  $t_{\text{now}}$ . *Laser* reads inputs #2, #4 and #24 from preceding modules, which are all at  $t_{\text{now}}$ , and uses them to calculate its output. *mirror1* then calculates outputs #7 and #17 using input #6 from “Laser”, and #16 from “prop2”. *PD* then calculates output #19 using the output of *mirror1*, thereby making data at  $t_{\text{now}}$  available to *out1*. *prop1* reads input #8 to update its internal state, as does *prop2*. Note that both of these modules have already produced output at  $t_{\text{now}}$ . Next *DF* calculates output #21 using input #19, and *delay* updates its internal state using this output.

LIGO-DRAFT

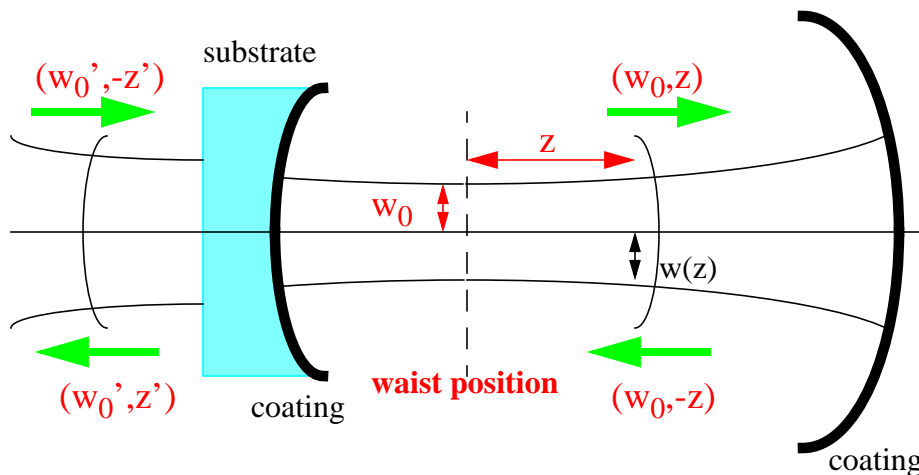
## 8 MODE SUPPORT

### 8.1. Modal model

Modal expansion using the Hermite Gaussian expansion is used to implement the alignment degrees of freedom and beam profile. Appendix 1 summarizes the mathematics needed for the modal model implementation. Same normalization will be used as the Modal Model Paper (1).

In the case of a circular Gaussian beam, the base of the modal expansion can be completely specified by two quantities, the waist size and the waist position. When a field representation changes from one set of a waist size and a waist position to another set, it is called “the base of the field is changed”. A field at a given location will carry the information of the representation base using the waist size and the distance to the waist position. The field going away from the waist position (diverging beam) will have positive distance, while that coming to the waist (converging beam) will have negative distance, just as is illustrated in Figure 6.

However, E2E supports elliptical Gaussian beams. The Gaussian beam solution is thus written in terms of a set of astigmatic Hermite-Gaussian modes by writing separate values of waist size and waist position for the horizontal X and vertical Y coordinates. The base of the Gaussian beam in E2E is thus represented by four quantities: “waist\_size\_X”, “waist\_size\_Y”, “dist\_waist\_X” and “dist\_waist\_Y”. To keep our following discussion simple we refer to just waist-size and waist position as the base of the modal expansion and note that these quantities for two transverse directions, X and Y, can change or evolve in an independent way.



**Figure 6: Time domain Modal Model field**

A field is represented by a base of the mode representation and a mode amplitude vector, which is a vector of amplitudes for each eigen state:

$$E(x, y, z) = \sum a_{mn} \cdot U_{mn}(x, y, z) \quad (1)$$

where  $U$  is defined in eqn. (A.3). In other words, by a set of  $U_{mn}$ , and weights of each mode. When the base is changed,  $U_{mn}$ 's change.

The simulation in Figure 6 will go as follows. From the left, a field based on a base B1 goes into a substrate, and the field shape changes by the lensing effect of the substrate. A new base B2 is calculated on the right hand side of the substrate so that the incoming TEM00 becomes a outgoing TEM00 in the new base, or, the new base B2 is chosen so that the incoming field  $a_{00}U_{00}^{B1}$  becomes  $a_{00}U_{00}^{B2}$  after the lens. The field then interacts with the coating, or mirror, and some are reflected back to the left and some are transmitted into the cavity. The base of the reflected field is changed back to B1. The field in the cavity, going back and forth, is represented using base B2.

As is explained in this case, the simulation can start from the source with arbitrary base, then propagates down to the core optics. In reality, optics are arranged in between the laser and the core optics so that the mode matches at the core optics. What this means is that, in the example Figure 6 and the discussion above, the new base B2 is, ideally, the eigen state of the mode of the cavity defined by the two mirrors. To help this situation, the E2E package will provide a way to define the mode at the input which will match to the mode of the cavity when it goes into the core optics. The mode mismatches that occur due to small misalignments are, of course, what the E2E package simulates.

## 8.2. Basic approach - step by step

### 8.2.1. Convention:

8.2.1.1 **distance:** negative for converging beam, positive for diverging beam.

8.2.1.2 **radius of curvature:**

8.2.1.2.1 *(for surface of mirror or lens) positive if looks concave from outside and negative if looks convex from outside*

8.2.1.2.2 *(for wavefronts of beams) positive for diverging beam and negative for converging beam*

8.2.1.2.3 *focal length: converging lens has positive value, diverging lens has negative value.*

$$\frac{1}{f} = -(n_{ref} - 1) \left( \frac{1}{R_1} + \frac{1}{R_2} \right)$$

### 8.2.2. Definition of fields

$$E(x, y, z, t) = \exp(i\omega_0 t) \cdot \sum_i \sum_{n, m=0}^{\max\_mn\_order} E_i^{mn}(z, t) \cdot U_{mn}(x, y, z) \quad (2)$$

$$E_i^{mn}(z, t) = \exp(i\Omega_i t) \cdot \tilde{E}_i^{mn}(t) \quad (3)$$

$$\tilde{E}_i^{mn}(t) = P_i^{mn}(z) \cdot \bar{E}_i^{mn}(t)$$

$$P_i^{mn}(z) = \exp(-i(k_0 + k_i)z) \cdot \exp(i(n + m)\eta_{00}(z)) \quad (4)$$

$$U_{mn}(x, y, z) = u_m(x, z) \cdot u_n(y, z) \quad (5)$$

$$u_m(x, z) = \left(\frac{2}{\pi}\right)^{\frac{1}{4}} \cdot \left(\frac{1}{2^m m! w(z)}\right)^{\frac{1}{2}} \cdot H_m\left(\frac{\sqrt{2}x}{w(z)}\right) \cdot \exp\left(-x^2 \left(\frac{1}{w(z)^2} + \frac{i(k_0 + k_i)}{2R(z)}\right)\right) \quad (6)$$

In these expressions, the index  $i$  runs for all sidebands and a carrier. As is shown below, each field component of a field carries its own frequency, and there is no restriction among them. Strictly speaking, the wave number  $k = 2\pi/\lambda$  is calculated as  $k_0 + k_i$ , where  $k_0$  is a reference wave number which is very close to or identical to that of the carrier, or  $k_i/k_0 \sim \lambda_{CR}/\lambda_{SB} \sim 10^{-7}$ . The angular velocity and wave numbers are related as  $k_0 = \omega_0/c$ , and  $k_i = \Omega_i/c$ . In the simulation program,  $\tilde{E}_i^{mn}(t)$  is kept as sub field components, and the phase change  $P_i^{mn}(z)$  is calculated by the propagator.

### 8.2.3. A choice of the mode base at the laser

The simulation starts from a laser, and the base of the field at the laser source can be anything. A support will be provided to define a base at the laser using the following strategy. A pair of mirrors forming a resonant cavity will be used to define one base of the HG expansion. Examples are, (1) two mirrors of the FP cavity, (2) one of the arm cavity of LIGO, or (3) the recycling mirror and the inline end mirror of the recycled Michelson cavity. It is assumed that the whole core optics part is designed to mode match, at least approximately. The support will provide the base which, after passing through optics between the laser and core optics with appropriate base changes, matches the base chosen using the resonant cavity defined above.

### 8.2.4. Propagator

The macroscopic length,  $L$  of the cavity minus a small length corresponding to the Guoy phase acquired during propagation of this length is assumed to be an integral multiple of wavelength (Note: This is just our choice of convention, although a half-integral multiple would have been sufficient to keep the TEM00 mode resonant inside the cavity). The propagator matrix, thus, just adds up Guoy phase to only higher order modes [i.e.,  $(m+n)G(z)$  and not  $(m+n+1)G(z)$  where  $G(z) = \arctan(Z/Z_0)$ ,  $Z_0 = \pi w_0^2/\lambda$  being the Rayleigh range] and the common longitudinal phase to all modes.

### 8.2.5. field is modified by mirror and lens and wedge

The base of Hermite\_Gaussian expansion can be represented by a complex number

$$q \equiv (z, -z_0)$$

This parameter is changed by mirror or lens in following ways:

#### 8.2.5.1 A lens changes the base of Hermite-Gaussian expansion.

$$q_{out} = \frac{q_{in}}{1 - q_{in}/f}$$

where f is the focal length of lens.

The lens may also mix the components. E.g., thermal lensing effects.

Choice of the origin of z is relative to the waist position where guoy-phase (for all modes) equals to zero.

#### 8.2.5.2 A mirror changes the mixture of components in the same expansion base.

- On reflection with normal incidence from a mirror,  $z \Rightarrow -z$  (by convention).
- On reflection with non-zero incidence angle from a mirror (different for bases in horizontal X and vertical Y direction, i.e., in the plane of incidence and perpendicular to plane of incidence respectively )

$$q_{outX} = \frac{q_{inX}}{1 - 2q_{inX}/(R \cos \theta)}$$

$$q_{outY} = \frac{q_{inY}}{1 - 2 \cos \theta q_{inY}/R}$$

- by substrate lensing effect (as described in sec. 8.2.4.1)

#### 8.2.5.3 Mirror and lens

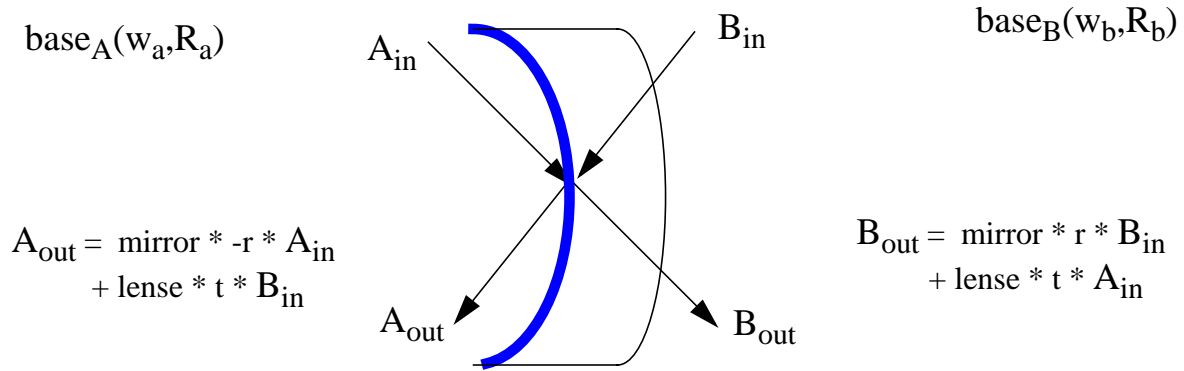
	<i>Lens</i>	<i>Mirror</i>
Input mechanical	<b>z</b> : small longitudinal displacement	<b>z</b> : small longitudinal displacement <b>pitch, yaw</b> : small angular rotation
Input optical	<b>field</b> : single or multi-mode	<b>field</b> : single or multi mode field

## 8.2.5.3 Mirror and lens

	<i>Lens</i>	<i>Mirror</i>
Output	<b>field</b> : affected by lens	<b>field</b> : affected by mirror and its lensing effect
Setting	<b>t, r</b> : transmittance and reflectance <b>radi_front, radi_back and refrac_index</b> : radii of curvature and refractive index	<b>t, r</b> : transmittance and reflectance. radi_front, radi_back and refrac_index : mirror curvature on coated side and non-coated (AR coated side) side and refractive index of dielectric Two sets of t,r for P and S polarization will be introduced later. Possibly explicit mode decomposition matrix, effectively representing some aberration
action	lens : change of HG base. When a non-zero field comes into the optic for the first time, check the consistency, i.e., the bases of two components of Aout in Figure 7 are the same.	mirror : mode decomposition and also the lensing effect, i.e., change of HG base of partial beams transmitted to both sides. When a non-zero field comes into the optic for the first time, check the consistency, i.e., the bases of two components of Aout in Figure 7 are the same.

LIGO-DRAFT

### 8.2.5.4 Schematic summary of optics action



- 1) Field A (B) is represented by using HG eigenstate base A (B).
- 2) mirror decompose components in the same base
- 3) lense changes the base between A and B, while keeping the mode components

**Figure 7: Schematics of optics**

### 8.2.6. Only lenses between COC and laser change the base

#### 8.2.6.1 Program to calculate the modal-base at the laser using the base defined by the pair of mirrors in COC

In the following, we consider a simple case: a laser and a Fabry-Perot cavity. However, the same step-by-step method can be easily extended to more complicated set-up.

- Rayleigh range for the modal-base inside the cavity formed by two mirrors (mirror 1 is near to laser) :

$$Z_0 = \frac{\pi W_0^2}{\lambda} = \sqrt{\frac{L(R_1 - L)(R_2 - L)(R_1 + R_2 - L)}{(R_1 + R_2 - 2L)^2}}$$

Where  $R_1$  and  $R_2$  are absolute values of radii of curvature of two mirrors. Distances of the waist from mirror 1 and mirror 2 are respectively:

$$Z_1 = \frac{L(R_2 - L)}{R_1 + R_2 - 2L} \quad Z_2 = \frac{L(R_1 - L)}{R_1 + R_2 - 2L}$$

- At the input mirror 1 and inside the cavity, the radius of curvature of the beam wavefront is, of course,  $R_1$ . The half-width of the beam is:

$$W = W_0 \left[ 1 + \left( \frac{Z_1}{Z_0} \right)^2 \right]^{1/2}$$

- At the input mirror 1 and just outside the cavity, the radius of curvature of the beam (due to the

lensing effect of mirror) is

$$R_{out} = \frac{R_1}{n}$$

where  $n$  is the refractive index of the mirror dielectric material (for fused silica,  $n=1.47$ ).

The change in the width size,  $W$ , after it passes through the mirror is assumed to be negligible.

- We are now ready to calculate modal-base for the laser from two known quantities: radius of curvature of beam wavefront,  $R_{out}$  and its width,  $W$  at the input mirror just outside the cavity. The waist-size for the new base and the distance of the (virtual) waist position from the input mirror are :

$$W_0^{(new)} = \frac{WK}{\sqrt{1+K^2}} \quad Z = \frac{R_{out}}{1+K^2}$$

where

$$K = \frac{\lambda R_{out}}{\pi W^2}$$

**8.2.6.2 multi-optics has a pure virtual function to convert the base. All distances are neglected expect for the configuration of COC**

**8.2.6.3 Any hardware between the laser and COC which need to take care of the spatial distance should be implemented as a separate module. A module, micro\_prop, may serve for some case which just advance the phase front by some specified amount.**

## 8.2.7. single and multi mode fields

One Field is specified by a reference wave number,  $k_0 = 2\pi/\lambda_0$ , and a set of subfields,  $(dk_i, E_i)$ . Subfields include both sidebands and a carrier field. The wave number of each subfield is  $k_0 + dk_i$ , and the amplitude of that subfield is specified by  $E_i$ . For the multi mode field,  $E_i$  is an array of amplitudes, each element of the vector representing the amplitude of each mode. The modes represented by the Hermite Gaussian  $U_{mn}$ , (A.3), are ordered in the following way:

$$(0,0), (1,0), (0,1), (2,0), (1,1), (0,2), \dots$$

The “reference field” is defined in field\_gen module. The “field” is characterized by the following quantities.

- Single **mode** or multi mode field by Hermite Gaussian expansion
- Reference field **wave length**
- **Power**( $P(t)$ ) and **phi**( $d\phi(t)$ ), separately for each mode for the multi mode field, with which the field is expressed as  $\sqrt{P(t)} \cdot \exp(i(\omega t + d\phi(t)))$ . Sidebands are generated later by other modules - sideband\_gen or phase\_adder.
- If the simulation is the mutil mode case, **total number of modes**, **waist size in horizontal X**

and vertical Y direction and the distance to the waist position in horizontal X and vertical Y direction are also needed.

- The “field” also carries other auxiliary options.
  - **how the mode decomposition is calculated**, either using full expression or using a smart algorithm, explained in another e2e manual, LIGO-T970196E.

User specifies these information by the inputs to the field\_gen module, which has two inputs and five settings.

- **Inputs**
  - Power(1,1.0) - array of real, specifying the power of each mode
  - phi(1,0.0) - array of real, specifying the phase fluctuation of each mode
- **Settings**
  - lambda (1.064e-6) - real, wave length of the reference field
  - mn\_order(1) - integer, specifying the order of transverse modes. Use mnorder = -1 to specify the single mode field.
  - waist\_size\_X, waist\_size\_Y(0.0) - real, the waist size
  - distance\_waist\_X, distance\_waist\_Y (0.0) - real, distance to the waist position
  - polarisation: p or s represented by 0 and 1 respectively.
  - field\_options (1) - integer, 0 for full algebra, 1 for smart algebra

For each, default values are also shown in brackets. For vectors, first number is the size of the

**Table 1: Field structure**

<i>Mode</i>	<i>Global</i>	<i>Component</i>
Single	$k_0$ : time independent wave number <b>update flag</b> : number of SB	$(\delta k, E)_i$ : wave number correction and amplitude for each RF sideband.
Multi	$k_0$ : time independent wave number $\max(m+n)$ : number of modes <b>mode decomposition</b> : fast or slow $(w_0, z)$ for both X and Y direction : parameters defining the Hermite-Gaussian base. $w_0$ is the waist size and $z$ is the distance to waist. <b>update flag</b> : number of SB, HG base <b>pol</b> : polarization status - P, S or mixed	$(\delta k, (E)_j)_i$ : For each sideband, the field amplitude represent a vector of amplitudes for each mode. mode is order in the following way : (00),(10),(01),(20),(11),(02),... or $j = \frac{(m+n) \cdot (m+n+1)}{2} + n$ When polarization is implemented, vector $(E)_j$ will be duplicated for each polarization status.

array and the rest represent the values.

From the user settings of these inputs, or using the default values if not specified explicitly, the field quantities are set in the following order.

1. The mode structure is decided by “mnorder”. When mnorder = -1, single mode field is simulated, and for non-negative mnorder, multi mode field is calculated including  $n+m \leq mnorder$ . For example, when mnorder = 2, the simulation will include modes with  $(n,m) =$

(0,0), (1,0), (0,1), (2,0), (1,1) and (0,2).  $mnorder = 0$  is supported mainly for the code validation purpose and  $mnorder = -1$  should run faster.

2. If the size of the Power or phase is less than the number of modes simulated, they are assumed to be 0. For example,  $mnorder = 1$ , and  $Power=[1.0]$ , then it is assumed that the initial power for TEM01 and TEM10 were 0.

### 8.2.8. Polarization

**8.2.8.1** At present the polarisation setting (p or s) in field is used just to determine the phase to be gained by a beam on reflection from the sides of a mirror. Capability of handling mixture of both polarization states has not yet been implemented.

8.2.8.2 Mirrors have separate r and t for each polarization. Only one set of values (corresponding to only one polarisation) can be set for the current implementation.

**8.2.8.3** Our final implentation will take care of the above two restrictions (i) by doubling the number of field amplitudes carried by the “field” class (ii) by having two sets of reflectivity and transmittivity values, each for one polarisation.

## 8.3. Detector, demodulation and shotnoise

### 8.3.1. Detector matrix

Using the expression defined in Section 8.2.2., the power observed by a detector located at a given z position is given by the following expression. In this calculation,  $\exp(-ir^2(k_0+k_i)/R)$  is approximated by  $\exp(-ir^2k_0/R)$ , because the correction is of the order of  $\lambda_{CR}/\lambda_{SB} \sim 10^{-7}$ .  $D(x,y)$  is a pupil-weighting function, and  $\Omega$  is the area of the photodetector. As can be seen from Eq.(7), a detector can be characterized by a detector matrix  $D_{mn,m'n'}$ .

$$\begin{aligned}
 P &= \int_{\Omega} dx dy |E(x, y, z, t)|^2 \cdot D(x, y) \\
 &= \sum_{m, n, m', n'} E^{mn}(z, t) \cdot E^{m'n'}(z, t)^{\dagger} \cdot D_{mn, m'n'}
 \end{aligned} \tag{7}$$

$$E^{mn}(z, t) = \sum_i \exp(i\Omega_i t) \cdot \tilde{E}_i^{mn}(t) \tag{8}$$

LIGO-DRAFT

$$\begin{aligned}
D_{mn,m'n'} &= \int_{\Omega} dx dy U_{mn}(x, y, z) \cdot U_{m'n'}(x, y, z) \cdot D(x, y) \\
&= d_{mn} d_{m'n'} \cdot \int_{\Omega} dx dy e^{-(\hat{x}^2 + \hat{y}^2)} H_m(\hat{x}) H_{m'}(\hat{x}) H_n(\hat{y}) H_{n'}(\hat{y}) D(\hat{x}, \hat{y}) \quad (9) \\
&= \frac{1}{2} d_{mn} d_{m'n'} \cdot \int_{\rho_{min}}^{\rho_{max}} d\rho \int_{\Phi_{min}}^{\Phi_{max}} d\phi H_m(\hat{x}) H_{m'}(\hat{x}) H_n(\hat{y}) H_{n'}(\hat{y}) D(\hat{x}, \hat{y}) \\
d_{mn} &= \frac{1}{\sqrt{\pi 2^{m+n} m! n!}} \\
\hat{r} &= \frac{r}{w(z)} \quad \hat{x} = \frac{\sqrt{2}}{w(z)} x = \hat{r} \cos(\phi) \quad \hat{y} = \frac{\sqrt{2}}{w(z)} y = \hat{r} \sin(\phi) \quad (10) \\
\rho_{min} &= \exp(-\hat{r}_{max}^2) \quad \rho_{max} = \exp(-\hat{r}_{min}^2)
\end{aligned}$$

In the simulation package, if the detector matrix cannot be calculated analytically using formulas in the Appendix, the matrix is calculated numerically. The D matrix, Eq.(9), is real and fully symmetric with respect to the exchange of m and m' and n and n', or

$$D_{mn,m'n'} = D_{m'n,mn'} = D_{mn',m'n} = D_{m'n',mn} \quad (11)$$

### 8.3.2. Demodulation

By substituting the explicit frequency dependence eq.(8) to eq.(7), the expression of the power in a detector can be rewritten in the following way.

$$\begin{aligned}
P(t) &= \sum_{i,j} P_{ij} \cdot \exp(-i(\Omega_j - \Omega_i)t) \\
&= \sum_i P_{ii} + \sum_{i < j} (P_{ij} \exp(-i\Delta_{ij}t) + \text{c.c.}) \quad (12) \\
&= \sum_i P_{ii} + 2 \sum_{i < j} (P_{ij}^{Im} \sin(\Delta_{ij}t) + P_{ij}^{Re} \cos(\Delta_{ij}t)) \\
P_{ij} &= \sum_{m,n,m',n'} \tilde{E}_i^{mn}(t) \cdot \tilde{E}_j^{m'n'}(t)^\dagger \cdot D_{mn,m'n'} = P_{ij}^{Re} + iP_{ij}^{Im} \quad (13) \\
P_{ij}^\dagger &= P_{ji} \quad P_{ij}^{Re} = P_{ji}^{Re} \quad P_{ij}^{Im} = -P_{ji}^{Im}
\end{aligned}$$

$$\Delta_{ij} = \Omega_j - \Omega_i \quad (14)$$

The CD part of the power is given by the first term in eq.(12),

$$P_{DC} = \sum_i P_{ii} \quad (15)$$

The conversion of the phase modulation adopted in this simulation is

$$E_{modulated} = E_{in} \exp(i\Gamma \sin(\Omega \cdot t)) \quad (16)$$

When the photo diode current is demodulated using a sinusoidal wave with the demodulation frequency of  $\Delta\Omega$ , the inphase and quad phase demodulated amplitudes are given as follows:

$$P_{In} = \frac{1}{T} \int_0^T P(t) \sin(\Delta\Omega \cdot t) = \sum_{\Omega_j - \Omega_i = \Delta\Omega} P_{ij}^{Im} \quad (17)$$

$$P_{Qu} = \frac{1}{T} \int_0^T P(t) \cos(\Delta\Omega \cdot t) = \sum_{\Omega_j - \Omega_i = \Delta\Omega} P_{ij}^{Re} \quad (18)$$

The module “pd\_demod” produces a complex value defined by the following equation.

$$P_{demod} = \sum_{\Omega_j - \Omega_i = \Delta\Omega} P_{ij} \quad (19)$$

So the imaginary part of the pd\_demod output gives the inphase demodulated output and the real part gives the quadphase demodulated output.

### 8.3.3. shot noise

Because of the quantum nature of the electric field, the measured current of the photo diode fluctuates around the expected value. For a given input field power  $P(t)$ , the average number of photons for a measured time interval  $\Delta t$  is given by the following formula

$$n_0(t) = \frac{e_0(t)}{h \cdot \nu} = \frac{\eta \cdot P(t) \cdot \Delta t}{h \cdot \nu} \quad (20)$$

where  $h\nu$  is the energy of a photon and  $\eta$  is the quantum efficiency of the detector. The measured number of photons fluctuates randomly following the poisson distribution with the given average value.  $\mathfrak{N}(n)$  is used to denote a number observed with the average value of  $n$ .

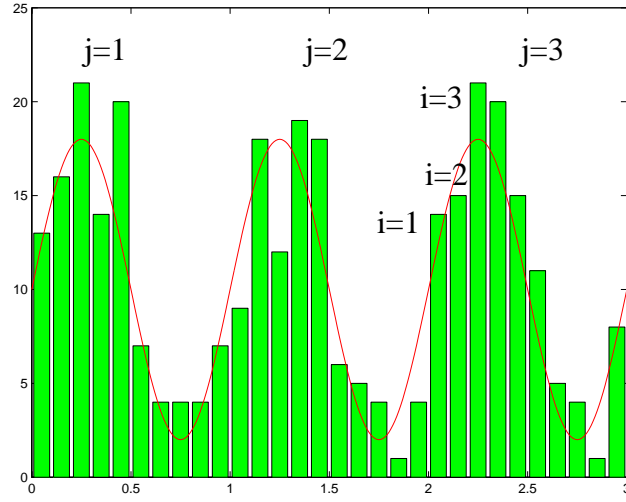
When a measurement is done for a time interval of  $\tau$  using a modulation frequency  $\Omega=2\pi/T$ , the power, inphase and quad phase demodulated signals are given by the following expressions, where the integration over time is replaced by a summation with a time step of  $\Delta t$ .

$$\bar{P}_x^{obs} = \frac{1}{\tau} \int_0^{\tau} w_x(t) P^{obs}(t) dt = \frac{1}{n_b n_c \Delta t} \sum_{i=1}^{n_b} \sum_{j=1}^{n_c} w_x(t_i^j) h\nu \aleph(n_0(t_i^j))$$

$$w_{DC}(t) = 1 \quad w_{In}(t) = \sin(\Omega_{demod} \cdot t) \quad w_{Qu}(t) = \cos(\Omega_{demod} \cdot t) \quad (21)$$

$$n_b = \frac{T}{\Delta t} \quad n_c = \frac{\tau}{T} \quad t_i^j = (i + n_b \cdot j) \Delta t$$

$n_c$  is the total number of oscillations in the observation period, and  $n_b$  is the number of binning



**Figure 8: Quantized power**

per oscillation. If the time dependence of the field, or  $P_{ij}$  in eq.(13), is neglected during the measurement period  $\tau$ ,  $\aleph(n)$  and  $w_x(t_i^j)$  are independent of the cycle, or  $j$ , and eq.(21) can be rewritten in the following way.

$$\bar{P}_x^{obs} = \frac{h\nu}{\tau} \sum_{i=1}^{n_b} w_x(t_i) \aleph(n_c \cdot n_0(t_i)) \quad (22)$$

$$t_i = \Delta t \left( i - \frac{1}{2} \right)$$

The module “pd\_demod” uses this formula to calculate the shotnoise when the full simulation option is chosen. This full simulation is slow because it needs several 10 random number generation. In order to speed up the simulation, a fast option is provided. When that option is chosen, the power is approximated by

$$P(t) = P_{DC} + 2(P_{-1,1}^{Im} \cdot \sin(2\Omega_{demod} \cdot t) + P_{-1,1}^{Re} \cdot \cos(2\Omega_{demod} \cdot t)) \quad (23)$$

calculate the summation (22) analytically and generate the fluctuation for the power, in and quad demodulated outputs independently. This is correct when there is only one sideband as the demodulation frequency, except that the correlation among the three outputs cannot be simulated.

LIGO-DRAFT

## APPENDIX 1 MODAL MODEL FORMULAS

[ shamelessly stolen from the famous modal model paper (1) ]

One dimensional Hermite Gaussian

$$U_m(x, z) = \left(\frac{2}{\pi}\right)^{1/4} \left(\frac{1}{2^m m! w(z)}\right)^{1/2} H_m\left(\frac{\sqrt{2}x}{w(z)}\right) \exp\left(-x^2 \left(\frac{1}{w(z)^2} + \frac{ik}{2R(z)}\right)\right) \exp\left(i\left(m + \frac{1}{2}\right)\eta(z)\right) \quad (\text{A.1})$$

**Table 2:**

<i>name</i>	<i>expression</i>	<i>size for LIGO</i>
mode-dependent Guoy phase shift	$\eta(z) = \tan^{-1}\left(\frac{z}{z_0}\right)$	
spot size	$w(z) = w_0 \sqrt{1 + \left(\frac{z}{z_0}\right)^2}$	
curvature of the phase front	$R(z) = z + \frac{z_0^2}{z}$	
Rayleigh length	$z_0 = \frac{\pi w_0^2}{\lambda}$	
waist size	$w_0$	

$H_m(x)$  is the Hermite polynomial of order  $m$ . The following relations are used repeatedly in the calculations which follow:

$$\int_{-\infty}^{\infty} U_m^\dagger(x, z) U_n(x, z) dx = \delta_{mn} \quad (\text{A.2a})$$

$$2xH_m(x) = H_{m+1}(x) + 2mH_{m-1}(x) \quad (\text{A.2b})$$

$$\frac{d}{dx}H_m(x) = 2mH_{m-1}(x) \quad (\text{A.2c})$$

$$\int_{-\infty}^{\infty} U_m^\dagger(x, 0) \frac{H_i(\sqrt{2}x/w_0)}{H_k(\sqrt{2}x/w_0)} U_k(x, 0) dx = \sqrt{\frac{2^i i!}{2^k k!}} \delta_{mi} \quad (\text{A.2d})$$

Eqn. (A.2a) is the orthonormality condition; eqns. (A.2b) and (A.2c) are recursion relations to be used to derive Hermite polynomials of any order, beginning with  $H_0(x) = 1$ .

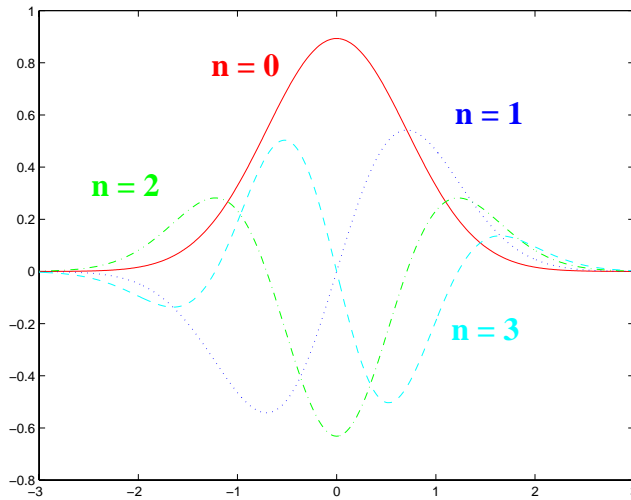
In two dimensions the Hermite-Gaussian modes are given by

$$U_{mn} = U_m(x, z) U_n(y, z) e^{-ikz} \quad (\text{A.3})$$

The explicit forms of a few low order Hermite polynomials are :

$$H_0(x) = 1; \quad H_1(x) = 2x; \quad H_2(x) = -2 + 4x^2; \quad H_3(x) = -12x + 8x^3; \quad (\text{A.4})$$

A few examples of the distribution of the Hermite Gaussians are shown below.



**Figure 9: HermiteGaussi**

The following formula is used in the modeling of the photo detector with simple shapes.

LIGO-DRAFT

$$\begin{aligned}
& \int_0^{\infty} H_n(x) H_m(x) \exp(-x^2) \\
&= \sum_{r=0}^{\lfloor \frac{n}{2} \rfloor} \sum_{s=0}^{\lfloor \frac{m}{2} \rfloor} (-1)^{r+s} \cdot \frac{n!m!2^{n+m-1}}{(2r)!!(2s)!!(n-2r)!(m-2s)!} \cdot \\
& \left( \begin{array}{l} \left(\frac{n+m-1}{2} - r - s\right)! \cdot 2^{-r-s} \text{ for odd } n+m \\ \frac{(n+m-2r-2s-1)!!}{\sqrt{2}^{n+m}} \sqrt{\pi} \text{ for even } n+m \end{array} \right)
\end{aligned} \tag{A.5}$$

**Table 3:**  $HH[m, n] = \frac{1}{\sqrt{\pi}2^{m+n}m!n!} \int_0^{\infty} H_m(x)H_n(x)\exp(-x^2)$

m \ n	0	1	2	3	4	5
0	$\frac{1}{2}$	$\frac{1}{\sqrt{2\pi}}$	0	$-\frac{1}{2\sqrt{3\pi}}$	0	$\frac{1}{4}\sqrt{\frac{3}{5\pi}}$
1	$\frac{1}{\sqrt{2\pi}}$	$\frac{1}{2}$	$\frac{1}{2\sqrt{\pi}}$	0	$-\frac{1}{4\sqrt{3\pi}}$	0
2	0	$\frac{1}{2\sqrt{\pi}}$	$\frac{1}{2}$	$\frac{1}{2}\sqrt{\frac{3}{2\pi}}$	0	$-\frac{1}{4}\sqrt{\frac{5}{6\pi}}$
3	$-\frac{1}{2\sqrt{3\pi}}$	0	$\frac{1}{2}\sqrt{\frac{3}{2\pi}}$	$\frac{1}{2}$	$\frac{3}{4\sqrt{2\pi}}$	0
4	0	$-\frac{1}{4\sqrt{3\pi}}$	0	$\frac{3}{4\sqrt{2\pi}}$	$\frac{1}{2}$	$\frac{3}{8}\sqrt{\frac{5}{2\pi}}$
5	$\frac{1}{4}\sqrt{\frac{3}{5\pi}}$	0	$-\frac{1}{4}\sqrt{\frac{5}{6\pi}}$	0	$\frac{3}{8}\sqrt{\frac{5}{2\pi}}$	$\frac{1}{2}$

$$\int_{-\infty}^{\infty} H_n(x)H_m(x)\exp(-x^2) = \begin{cases} \sqrt{\pi}2^n n! & \text{for } n=m \\ 0 & \text{for } n \neq m \end{cases} \quad (\text{A.6})$$

Other useful formulas.

$$H_n(x) = \sum_{r=0}^{\lfloor \frac{n}{2} \rfloor} (-1)^r (2r-1)!! {}_n C_{2r} 2^{n-r} \cdot x^{n-2r} \quad (\text{A.7})$$

$$H_n(x+y) = \sum_{r=0}^n H_{n-r}(x) 2^r {}_n C_r \cdot y^r \quad (\text{A.8})$$

LIGO-DRAFT

## APPENDIX 2      REFERENCE

3. Y.Hefetz, N.Mavalvala and D.Sigg, "Principles of calculating alignment signals in complex resonant optical interferometers," J. Opt. Soc. Am. B **14**, 1597-1605 (1997).

LIGO-DRAFT