LIGO-T2000311-v1

SIS20 - Overview

Alba Romero (IFAE), Hiro Yamamoto (Caltech)

This is an internal working
note of the LIGO project.

Contents

1 Introduction

This document is a guide on the "Stationary Interferometer Simulation", referred to as SIS hereafter, an FFT-based simulation package to calculate fields in a stationary interferometer of various configurations. SIS was originally developed to design the optical configuration of the Advanced LIGO Interferometer. SIS, among other simulation packages such as DarkF, Oscar or MIT-FFT, calculates fields in optical systems by taking into account realistic details of optical components, like macroscopic shapes and microscopic surface phasemaps.

Fields are calculated under static or stationary conditions, and the role of SIS is complimentary to frequency domain simulations tools like Finesse and MIST, and time domain simulation tools like e2e and Siesta. However, SIS does much more than that. It also serves as an analysis tool of optical system characteristics. For instance, after stationary state fields are calculated, mode analysis can be done to see the mode matching and the Gaussian fitting can be done to calculate effective beam size and curvature.

SIS was originally developed using C++, based on an object oriented architecture. SIS then integrated a FFT-based simulation package known as FOG (Fast Fourier Transform Optical Simulation of Gravitation Wave Interferometer), designed and developed by Richard Day (then working at EGO, r.arthur.day@gmail.com). FOG was written using matlab. R. Day analyzed the details of the FFT-based simulation framework and improved the acceleration algorithm of field calculations in complex optical systems together with G. Vajente (vajente@caltech.edu). SIS was fully rewritten in matlab when the original SIS package and FOG were integrated into one simulation environment.

SIS is like a toolbox for building optical systems. This toolbox consists of elements to build an IFO simulation setup, and of algorithms to combine these parts for an arbitrary configuration. By combining tools, any optical configuration could be built and simulated. However, the user has to assemble parts to build the setup for a specific optical configuration and various parameters have to be calculated and set by hand. SIS consists of three parts, building optical systems, setting up fields, and analyzing them.

The most common optical systems that we will be dealing with as examples in this document are Fabry-Perot cavities, as the optical configuration of aLIGO/AdVirgo is a dual recycled Michelson cavity with two long FP cavity arms. The Michelson cavity may be a simple short cavity, which is intrinsically unstable, or may contain a mode changing folding mirrors to make the cavity stable.

The details of the optics can be included in the simulation, just as we will detail afterwards. AOS components, like TCS and baffles, can also be included. Any measured optics data in common data formats can be used in the simulation without any prior conversion to a specific format.

SIS uses the change of the round trip phase in a cavity as the default locking algorithm. This is close to the locking using the Pound-Drever-Hall signal. When there is more than one cavity in the system, each cavity is locked by using algorithm close to the real experimental process. These algorithms can be overridden by implementing a derived class with proper methods.

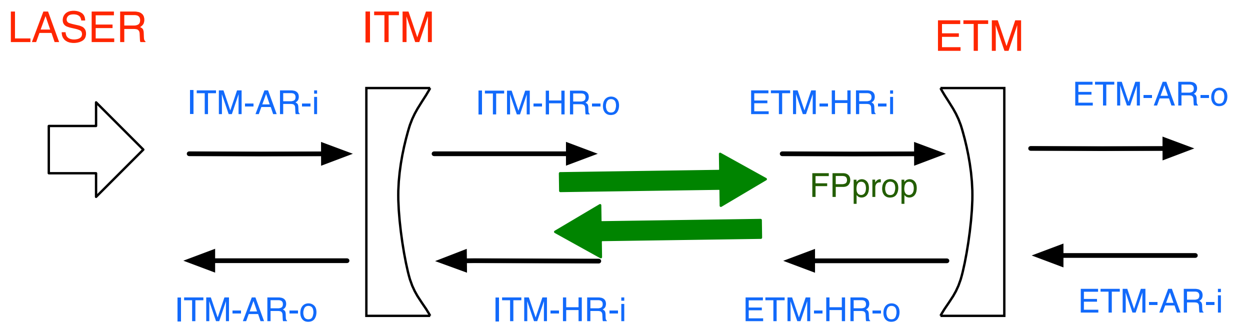
This document package consists of four documents.

1. **SIS20 - Overview** is an introduction of SIS with simple examples. Basic information on how to use SIS is covered in this document.
2. **SIS20 - Physics** explains formulations used in the SIS package, and there will be links quoted to related simulation codes.
3. **SIS20 - Programming** explains the details of the code structure, including the object structures and classes which are necessary to override some basic behavior.
4. **SIS20 - Applications** consists of practically useful examples. Each example will come with matlab codes.

We highly encourage the reader to have a look at the acronym list since we will be using them profusely throughout the note.

2 Simple Code Example

To use SIS, there are two steps. One is the definition of the IFO, and the other is the analysis of the fields in the IFO. A simple case example is shown below, **FPIFOAcc.m** to define a FP, and **runFPIFO.m** to analyze the field.



2.1 FPIFOAcc.m : definition of IFO

This IFO definition builds a FP cavity consisting of a laser source and two mirrors, named ITM and ETM.

```
classdef FPIFOAcc < FFTIFOAcc
    methods
        function obj = FPIFOAcc( varargin )
            obj@FFTIFOAcc( varargin{:} );
        end
        % overriding defineIFO to specify the IFO
        function defineIFO( obj, varargin )


            % basic FP parameters
            ATM = 0.326; Lcav = 3994.5; RoCITM = 1934; RoCETM = 2245;
            % ETM specifications
            delE = 0;  tiltX = 0;

            % all variables defined above this line can be changed as runtime parameters
            % fp = FPIFOAcc('tiltX', 1e-7);
            eval( UtilTK.setVars( who, true, varargin{:} ) );

            %% add optics
            obj.addLaser('LASER');
            obj.addMirror('ITM','invRoC',1/RoCITM,'Aperture',ATM,'T',0.014,'Thick',0.2);
            obj.addMirror('ETM','invRoC',1/RoCETM,'Aperture',ATM,'T',5e-6 );

            %% define connections of optics
            obj.addProp('','LASER','ITM-AR'); % link laser and AR side of ITM
            obj.addProp('FPprop','ITM-HR','ETM-HR',Lcav); % HR sides of ITM and ETM

            %% Mirror surface specification
            mapFile = '';                  % data file of the surface phasemap
            surfaceSpec = 'map + tiltX*x'; % any matlab code
            realRoC = RoCETM+delE;     % as measured RoC
            obj.setHRfiles('ETM',mapFile,surfaceSpec,0.16,realRoC,0,'tiltX',tiltX);

        end
    end
end
```

The red section defines the FP cavity and the green section defines the parameters. The blue section helps to let these parameters be changeable as runtime variables without modifying the code. (See Sec.2.2 for an explicit example.) The rest of the code shown in black is to make the IFO definition be part of the SIS object class definition, so just copy from the line with **classdef** to the last **end** line and modify the definition of the IFO or the code of **function defineIFO**.

The surfaceSpec argument in the setHRfiles specifies the actual surface shape. It can be any matlab expression, with several predefined variables, **map** is the surface map defined by mapFile, **x** and **y** are coordinates and **rsq** $= x^2 + y^2$. New variable names can be used in the surfaceSpec expression, and their values can be appended to arguments of setHRfiles, like the pair of 'tiltX' and tiltX in the example code above.

The mode of the input laser is chosen, by default, to match with the FP cavity mode calculated using the **design parameters**. To specify the mode explicitly, member function **setupBaseMode** needs to be overriden.

2.2 runFPIFO.m : analyze fields in FP cavity using FPIFOAcc.m

The following code, runFPIFO, uses this FP definition to study fields in a FP cavity.

```
% get field
% FP with small tilt
fp = FPIFOAcc('tiltX', 1e-6);
```

```
% lock the cavity
fp.lock;
% get the field reflected by ETM
% total field
[E,x] = fp.getField('ETM-HR-o');
% deviation from undisturbed field
[dE,x] = fp.Non00Amp('ETM-HR-o', false);
subplot(2,1,1)
semilogy(x, abs(E(end/2,:)).^2, x, abs(dE(end/2,:)).^2 );
line([0.163,0.163], [1e-10,1e5])
legend('Total power', 'Disturbance by tilt', 'mirror radius', 'location', 'sout
xlabel('x axis (m)')
ylabel('Power (W/m^2)')
title('Power density on ETM in a FP cavit with ETM tilt')

% HG(1,0) vs ETM tilt angle
tilt = (0:10)*1e-8; delE = -20:20:20;
HGC = zeros(length(delE),length(tilt));
for idelE = 1 : length(delE)   % different as-built ETM RoC
    for itilt = 1:length(tilt) % tilt angle scan
        % build IFO
        fp = FPIFOAcc('tiltX', tilt(itilt), 'delE', delE(idelE));
        % lock the cavity
        fp.lock;
        % analyze the field
        HGC(idelE,itilt) = fp.HGCoef('ETM-HR-o',1,0);
    end
end
subplot(2,1,2)
plot( tilt, abs([HGC(1,:); HGC(2,:); HGC(3,:)]).^2 );
title('HG10 power vs ETM tilt angle');
legend('dETMRoC = -20m', 'dETMRoC = 0', 'dETMRoC = +20m', 'location', 'northwes
xlabel('ETM Tilt angle')
ylabel('HG10 power')
```

There are three steps in the analysis code.

1. Build an IFO, **`fp = FPIFOAcc(runtime parameters);`**
2. Lock the IFO, **`fp.lock;`**
3. Analyze fields in the cavity, **`[E,x] = fp.getField(field specification)`** or **`HGC = fp.HGCoef(field specification)`**.

At the first step, the optical configuration of the IFO is analyzed and recognized, and necessary setups for the fast field calculations are prepared. The mode base is chosen either by **`FFTIFO.setupBaseMode`** defined by the user together with defineIFO, or by using a most likely cavity mode, like FP in this example. Parameters used in the IFO setups can be changed by using the runtime arguments. The example sets the tilt value and the deviation of the actual RoC by calling FPIFOAcc with pairs of arguments, ['tiltX', value of tilt] and ['delE', value of RoC change]. For example, if ['ATM', 0.5] is passed as the argument, a FP with aperture of 0.5m can be simulated.

After the first step is completed, the following line is printed, which shows the chosen mode base.

```
Cavity mode defined using field 'ITM-HR-o' with
w = 0.0529939, RoC = -1934, z = -1834.22, z0 = 427.807
```

At the second step, fields in a FP cavity are calculated using FFT, and the microscopic cavity length is adjusted to lock the cavity. Details of the field calculation and locking algorithm are discussed in the companion note, SIS-2-Physics. The function **calc** calculates fields with fixed cavity lengths. **lock** repeatedly calls **calc** with different cavity lengths to find the optimal locked length, based on the algorithm close to the PDH method.

During this locking process, the following lines are printed, which shows the convergence progress.

```
41 (FFT 328) :  'ITM-HR-o' Pwr= 2.697e+02,
Err= 2.080e-06, phiLoop= 3.385e-08, kL-gouy=-3.701e-02, delL= 8.597e-16
```

Each line shows the power of a field ('ITM-HR-o') in a cavity being locked. The `error` is the change of the field amplitudes between sequential iterations, and this is used as a criteria of convergence.

The laser frequency is defined by the default value for $\lambda = 1064\mu m$ [1] plus RF frequency, which is 0 by default. After locking a cavity, the RF frequency can be changed by **fp.setRF(RF)**. By calling **fp.setRF(RF)** and **fp.calc**, the frequency scan can be done under the condition that the FP is locked by CR or green laser.

Once the cavity is locked, the resonating cavity can be analyzed using various functions. (See Sec.4 and SIS-3-Programs for further reference.) Fields can be referenced by name based on the optics and the field direction, like `'ETM-HR-o'` or `'ETM-HR-i'`, a field going out from or coming into the HR side of ETM (see 4.3.1). The serial number, and other useful information about fields, can be found by fp.fields.printAll (See Appendix C).

[E,x,y] = fp.getField('ETM-HR-i') returns the complex amplitude E(j,i) at (x,y) = (x(i), y(j)) [2]. `Non00Amp` returns the difference between the resonating field E and the field without tilt, i.e., the disturbance by the aberration.

This example also calculates the coefficient of HG(1,0) mode at various tilt angles. The loop shown in this example shows how to use the runtime argument to simulate IFOs with different setups without modifying the IFO definition file.

[1] The default laser wavelength is defined by ConstantTK.lam0, and can be changed by addLaser('lamCR', CR wavelength).

[2] For now, fields are calculated in a symmetric square, so x and y coordinates are the same.

3 Overview of the SIS architecture

In the framework of SIS, the optical configuration is defined in a function **defineIFO** prepared by the user [3]. Main elements in defineIFO are **addMirror**s, which define optical elements, **addProp**s, which define connections of optical elements, and **setHRfiles**s, which define the details of the optical elements. addMirrors and addProps specify *design parameters*, and setHRfiles specifies the *actual parameters*. These parameters are explained later in this section.

All functions have additional parameters which can be used to customize each element. E.g., the aperture of a mirror can be specified by a keyword 'Aperture'. Source codes of SIS have help texts next to each function where all keywords are explained.

The first step to use the SIS package is to create the IFO object, like **ifo = FPIFO;** where FPIFO is the IFO definition setup by the user, like the code above. Example codes are provided with this document set.

SIS analyzes the IFO definition in defineIFO to recognize optical elements and connections among them. A connection corresponds to a field propagation. Fields are assigned to the source and sink of all propagation, like **ITM HR** $\rightarrow$ **ETM HR**. The field name going out of an optic has a postfix **-o** and one going into **-i**. The propagation **ITM HR** $\rightarrow$ **ETM HR** connects **ITM-HR-o** and **ETM-HR-i**.

After the IFO optical configuration is recognized, a mode is assigned to the laser and that mode propagates to all fields using **design parameters**, like RoCs (radius of curvature) specified in addMirrors. The laser mode can be provided by the user, and most appropriate one is assigned as default. For a FP cavity, the default laser mode is the one which matches to the FP cavity mode defined using design values of RoC of ITM and ETM and the cavity length in addProp, together with the static lens effect in ITM.

The mode analysis of the entire IFO is done to setup various parameters necessary for the FFT-based calculation. SIS calculates fields using an FFT based technique, and it does not use the modal model to calculate fields. The field calculation uses **actual parameters** and aberrations specified in setHRfiles.

One of the important parameters calculated using the mode analysis is the beam size on each optic. FFT-based calculation uses fields on fixed number of grid points (Nfft) in a finite area, like 128 x 128 points in 40cm x 40cm area. The area, FFT window size (Wfft), is chosen to be large enough to cover the optic of interest, and the resolution, Wfft / Nfft, needs to be small enough to recognize the variation of the field and the aberration structure of optics.

When the modal parameters for all fields are calculated, the optimal values of Wfft and Nfft are calculated. The number of grid points, Nfft, does not change from optic to optic, and Wfft is adjusted proportionally to the beam size on each optic. E.g., the beam size on aLIGO PR2 is 6.2mm and that on PR3 is 54mm, and the Wfft is scaled proportionally to the beam size, the scaling being 8.7 for this case. The field propagation is done taking this scaling into

[3]This member function of FFTIFO, discussed in Sec.5, is overridden to define a specific optical configuration.

account. Details are explain in the section **adaptive grid size**. After the window sizes are chosen, one value of Nfft is calculated so that the resolutions are enough at all optics. The default resolution is 1/16 of the beam size. E.g., the beam size on aLIGO ITM is 5.3cm, and the resolution is smaller than 3.3mm. When the user specifies resolutions for some optics, these constraints are included to calculate the most tight constraint to specify Nfft.

After Nfft and all Wffts are chosen, transmission, reflection and propagation maps are calculated together with other support parameters, using all information of the optical setup. These maps are Nfft x Nfft matrixes, and incoming fields are multiplied by these matrixes to calculate outgoing fields. These maps are calculated using actual parameters specified in setHRfiles [4]. When the surface aberration is specified in setHRfiles, the reflection matrix has this information included, and the field is disturbed when the incoming field is multiplied by the reflection map.

These preparations are done when the IFO object is created by the matlab command **ifo = FPIFO**. At this stage, all ingredients including placeholders of fields are prepared, but fields are empty.

After an IFO object is created, the fields in the IFO are calculated by using the built-in functions **lock** or **calc**. **calc** calculates stationary fields using the given cavity lengths. **lock** calls **calc** repeatedly to find the locked state of the IFO by adjusting various cavity lengths. The default method of the locking mimics the PDH technique without using side band fields. The details of these processes are explained in the following sections about member functions **FFTIFO.calc** and **FFTIFO.lock**, together with explanations in the note **SIS_2_Physics**.

After the IFO is built, **ifo = FPIFO**, and the cavity is locked, **ifo.lock**, the cavity can be analyzed by using built-in functions. There are many functions prepared to analyze the results of the fields calculated by **calc** or **lock**. Some examples are discussed in the following section.

4 Main Function List

4.1 Overview

In this section, several functions for analyzing fields and cavities are explained which are useful for many applications. A full list and detailed explanations of these functions and usage are given in the companion note, **SIS-3-Programs**, and actual applications are explained using more realistic simulation codes in **SIS-4-Applications**.

There are several kinds of functions.

1. Functions used to build an IFO optical configuration, like **`addMirror`** to add a mirror to IFO.
2. Functions used to analyze fields, like **`getField`** to get the field amplitude.

[4]There are two more functions, setARfiles and setTRfiles to specify details of the AR surface and the transmission.

3. Functions used to analyze cavities, like **`roundtripLoss`** to calculate the round trip loss of a cavity.
4. Functions used to study more details of IFOs, like **`getPointScatter`** to study the effects of point scattering.
5. Functions to analyse fields using mode expansions, like **`HGCoef`**. Modal model and FFT-based field analysis are complementary. Especially, when a few modes are dominant somewhere in a cavity, the mode analysis is very useful to understand what is going on.
6. Functions to manipulate data, like **`zernikeCoef`** which removes tilt or power or any zernike terms in a two dimensional data or **`loadOneDataFile`** which loads various kinds of data file.

Most of these functions are implemented as member functions of FFTIFO. Many of them are front-end of functions implemented in various class objects. FFTIFOAcc is a derived class of FFTIFO customized to use the acceleration of field calculations by R.Day and G.Vajente.

4.2 Runtime arguments : `ifo = IFODef('name1', val1, ...)`

The line **`eval(UtilTK.setVars(...))`** in **`FFTIFO.defineIFO`** makes it possible to specify all variables defined above this line by the runtime arguments. When the argument of the IFO specification has a pair of 'name' and *value*, e.g., **`fp = FPIFOAcc('tiltX', tiltVal)`** in the example runFPIFO in Sec.2.2, the variable specified by the '*name*' is changed to *value*. As this code example shows, one can calculate multiple cases using matlab loops, instead of modifying the code itself.

The variable name in the runtime argument does not need to be the full name of the variable. E.g., to specify a variable *'resolution'*, the runtime input name can be *'res'*. But the shortened name should uniquely identify one variable. If there are variables *RoCITM* and *RoCETM*, the runtime specification cannot be *'RoC'*, but it should be either *'RoCI'* or *'RoCE'*. When '?' is passed as the runtime argument, all variables and current settings are printed.

4.3 Conventions

4.3.1 Field and port naming

To specify a field, we use either a name like **ITM-HR-o**, or a serial index number, like **11** [5]. Field names are based on the optics names specified when addMirror is called. The name of the field on the HR (AR) side is appended by -HR (-AR) and the outgoing (incoming) direction is appended by -o (-i). E.g., for the outgoing field from the ITM on the HR side we have: **ITM-HR-o**. For an optic with which fields interact with finite incident angle, the HR (AR) has suffix of x and y, like **BS-HRx-i**. The port on an optic has the same name as the field which is connected to the port.

[5] index=ifo.fields.findNamedField('name') or use ifo.fields.printAll.

4.3.2 Axis

For a field, the $+\mathbf{z}$ is defined to be the propagation, and for an optic, it is defined to be the direction perpendicular to the mirror surface. The $+\mathbf{y}$ direction is defined to be upwards, and the $+\mathbf{x}$ direction is defined using $+\mathbf{y}$ and $+\mathbf{z}$ directions. Because of this convention, the direction of $+\mathbf{x}$ changes when a field is reflected.

When the mirror rotation is defined by $optPhi$ and $optTheta$ in addMirror, the mirror is first rotated around the z axis by $optPhi$, and then it is rotated by $optTheta$ around the new y axis. To setup a pitch tilt, use $optPhi = \pi/2$, $optTheta = pitch\ value$, and to setup a yaw tilt, $optPhi = 0$ and $optTheta = yaw\ value$.

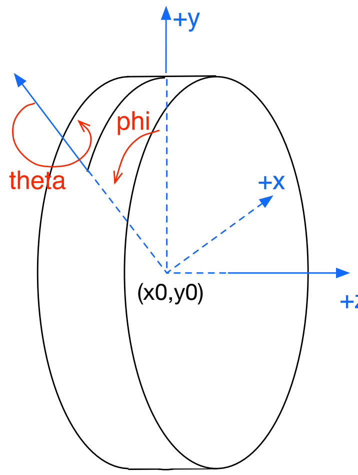


Figure 1: Axis definition

4.3.3 Function calls and Default values

Some of the functions, like addLaser, are used to build IFOs and they are used in building the FFTIFO object, like in defineIFO as shown in Sec 2.1. In there, the object to which the function belong to is obj, and the functions are references as obj.func, like obj.addLaser. Other kind of functions, like getField in Sec 2.2, are used to access data of the built IFO object, like ifo=FPIFOAcc, and the functions are referenced as ifo.func, like ifo.getField.

To use functions of SIS, the calling convention is as follows.

```
 [outVal1, outVale2, ...]  =
function(inVal1, inVal2, ..., 'name1', val1, 'name2', vale2, ...)  ;
```

addLaser is used to add a laser to the IFO in defineIFO, and it is called as follows.

to add a laser named 'LASER', power of 1W, laser wavelength of $1\mu m$

```
ID = obj.addLaser( 'LASER', 'power', 1, 'lamCR', 1.064e-6 );
```

getField returns a field amplitude of a specified field.

to get the field going to ETM, within a window size of W = 0.8m, propagation distance L = 3900m from the source

```
[E, x] = ifo.getField( 'ETM-HR-i', 0.8, 3900);
```

Several input arguments are mandatory to be specified in the specified order. Optic name in addLaser and field ID in getField are mandatory and to be given as the first input, but all the rest are optional.

For those optional input arguments, when the arguments are given by a pair of {'name string', value}, the ordering is arbitrary. So the input specification by 'power' and 'lamCR' can be reversed, or can be completely omitted. For getField, the window size and the propagation distance are optional, but the ordering is fixed.

When arguments are not specified, reasonable values are assigned as default values. E.g., the default values of power and lamCR are 1W and 1.064μm. The modes can be specified in addLaser, and default values are 00 mode which matches with the main IFO. The window size in getField is set based on the beam size and the mirror size, and the default propagation distance is the location of the field.

When a variable has a choice of values, like field mode being 'Hermite Gauss (HG)' or 'Laguerre Gauss (LG)', those choices are shown as {**`val1, val2, ...`**}, and when a default value is assigned, like the input power being 1W, it is shown as **`[default value]`**.

To see the full argument list of functions in the installed SIS, use the help command as follows.

`help ifo.addMirror` or **`help FFTIFO.addMirror`**

4.4 Mode base

The mode base to calculate various quantities are defined using the **design parameters** of optics, which are specified in addMirror('invRoC') and addProp(length specified as the last argument).

4.5 printAll

Main information about SIS data can be shown by using **`ifo.printAll('fileName')`**. This prints out many information of the SIS simulation setup in a readable format. If no fileName is specified, the information is displayed on the console window. This information includes an list of fields, propagators and cavities. ABCD matrixes of all optics and propagators, gouy phases of the propagators and losses at various places are also calculated and printed. If you specify the object name, like `ifo.`**`fields`**`.printAll`, only information about that object is printed.

The full output of **`fp.printAll('FPIFOAcc.txt')`** is included in the source code repository, and a part of the output is included in Appendix C with some comments.

4.6 building IFO

- **`ID = addLaser(name,'power',power,'lamCR',lamCR,`**
  `'fieldMode', mode, 'ind1', ind1, 'ind2', ind2);`
  - This function adds a laser source to the IFO, and returns the serial number of this laser.
  - The user can specify the laser power [1W], the wavelength of the light [1.064μm] and the injected mode ['HG'], 'LG' with two indexes (m,n) for HG case and (l,m) for LG case [0,0].
  - If the mode base is not specified, the input mode is chosen to match with the IFO mode defined by the design parameters.

- **ID = addMirror('mirName',**
  'invRoC', invRoC, 'Aperture', A, 'Thick', d,
  'T', T, 'Lhr', Lhr, 'Lar', Lar,
  'removeCurvature', remCur, 'removeTilt', remTil,
  'incAngle', incA, 'resolution', res,
  'optTheta', thetaI, 'optPhi', phiI);
  - This function adds a mirror named **'mirName'** to the IFO, and returns the serial number of this mirror.
  - The mirror shape is specified by **'Aperture'**, **'Thick'** and **'invRoC'**, inverse of radious of curvature of HR side.
  - Optical quantities are specified by the power transmittance **'T'** and HR and AR side power loss, **'Lhr'** and **'Lar'**. The reflectance is calculated using the transmittance and loss.
  - When **'removeCurvature'** is true, the RoC is changed so that the effective curvature seen by the design field is the design curvature, i.e., same role as a ring heater.
    A field of TEM00 mode with the design mode (RoC and w) is injected onto the mirror at the normal direction, and the amplitude of LG10 mode in the reflected field is calculated. The RoC of the mirror is internally adjusted to make this LG10 mode to be nullified. The value of the curvature change is stored in optDCurv of the mirror. The RoC of the mirror is $1/(mirror.invRoC + mirror.optDcurv)$.
    Run with **'removeCurvature' = true** is equivalent to run with **'removeCurvature' = false** and with **surfaceShape = 'map + rsq/2*optDCurv'** in setHRfiles using the optDCurv value to be the one calculated using **'removeCurvature' = true**.
    When the resonating cavity axis is tilted and the beam center hits off centered on the mirror, the TEM00 is injected along this resonating axis, and the reflected LG10 mode is calculated along this axis.

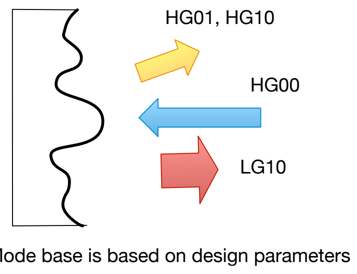


Figure 2: Remove curvature and tilt

  - When **'removeTilt'** is true, the mirror orientation is changed to serve as the alignment control using WFS. Calculation is done in the same way as 'removeCurvature'. The value of optTheta and optPhi of the mirror are changed to nullify the HG10 and HG01 modes in the reflected field.

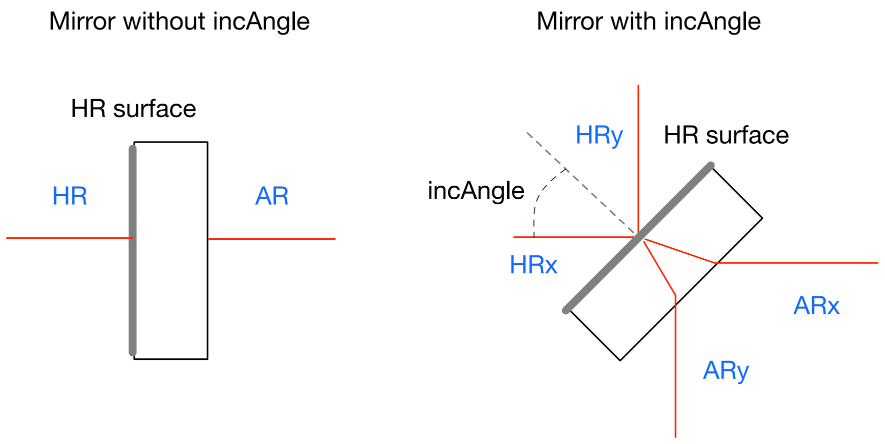


Figure 3: Port naming and incident angle

- When **'incAngle'** is specified, this mirror behaves like BS or RM2, i.e., the incident direction and the reflected directions are not parallel, and two directions are signified by x and y. See Fig. 4. Even when the specified value for 'incAngle' is 0, the mirror has two separate directions on each side.
- When **'resolution'** is not specified, the grid side of the FFT binning is chosen to be around 16th of the gaussian beam size. For aLIGO ITM, the beam size is 5.3cm and the bin size on ITM is ∼3mm. This grid size needs to be smaller than the structure of interest by factor of several. [6] Also this grid size affects the scattering angle. [7]
- **'optTheta'** and **'optPhi'** specify the mirror orientation defined in Sec.4.3.2. [8] Two functions are useful to utilize the orientation.
  * **[thetaI, phiI, thetaE, phiE] = cavityManager.beam2tilt(posIX, posIY, posEX, posEY, 1/ITMRoC, 1/ETMRoC, Lcav)**
    beam2tilt calculates the ITM and ETM mirror orientation from the beam positions on ITM (posIX, posIY) and ETM (posEX, posEY).
  * **obj.tiltOn('ITM')**
    When this is called, the input laser normal to ITM AR surface is bent to the direction of the resonating cavity axis and the beam is internally shifted to the beam position on ITM. In other words, the input laser beam is tilted and shifted so that it best matches with a FP cavity with tilted beam axis.

- **ID = addProp('propagatorName', sourcePort, sinkPort, length);**

[6] To simulate the scattering by the infamous spiral pattern of the LAM coating with spatial spacing of 8mm, the bin size was chosen to be 1mm.

[7] scattering angle $\sim \frac{laser\ wavelength}{spatialwavelength}$

[8] Don't turn on removeTilt when the mirror orientation is specified.

  - This function defines a connection of ports [9], from sourcePort to sinkPort, e.g., 'ITM-HR-o' to 'ETM-HR-i'.
  - length is the distance between the source and the sink of the propagator.

- **setHRfiles('mirName', 'mapFile', 'surfaceSpec', apertureRoC, measuredRoC, orientation, 'name1', val1, ...)**
  - This defines the details of the HR surface of the mirror 'mirName'.
  - 'mapFile', apertureRoC, measuredRoC : mapFile is the name of the HR surface data file. Various popular formats are supported. To support a new file format, modify the file **loadOneDataFile.m**. From the map data, the power term and tilt are removed using the map data in a circle defined by apertureRoC, and the measured RoC is added by using the measuredRoC with $r^2/(2 * measuredRoC)$.
  - **orientation** specifies the actual orientation angle in units of 90° of the installed mirror relative to the measurement. E.g., if the mirror installed is 90 rotated relative to the measurement, set orientation to be 1.
  - 'surfaceSpec' specifies the mirror surface shape by using expressions supported by matlab. The typical form is 'map + func(x,y,r)'. Here, map is the mirror surface data defined by 'mapFile', radius of curvature and tilt are removed. func can be any matlab function, either matlab functions like sin(x) or user provided functions, listed below. There are multiple functions prepared to be used in this surfaceSpec.
    * **calcThermal(elph, r, Psub, Pcoat, w, thickness, radius)** This is the coating thermal deformation using the Hello-Vinet analytic calculation [1]. **elph** is the choice of surface deformation (0) or thermal lens (1). **Psub** is the total absorption in the substrate, **Pcoat** is the total absorption in the coating, and w is the beam size on the mirror. **thisckness** and **radius** are the thickness and raius of the mirror. **help HVformula** shows detail explanation.
    * **pointAbs_SINB(x, y, x0, y0, P_abs, absRadius, testmass, ...)** This is the surface deformation by point absorbers using "Semi-Infinite model solution with Nonlinear Boundary condition" developed by Wenxuan Jia. Any number of PAs can be specified. E.g., with settings **x0=[-0.01, 0.02]; y0=[0.01, -0.03]; P_abs=[0.01, 0.03];** two absorbers are added at (-0.01,0.01) and (0.02,-0.03) with absorption power of 10mW and 30mW each. **help pointAbs_SINB** sgiws detail explanation.
    * **mapping(x, y, dat, newX, newY)** This maps the input 2D data using a new coordinate specified by newX and newY. With this function, the map data can be scaled or rotated.
  - **'name1', val1, ...** When the surfaceSpec needs variables, like the tiltX in the example FPIFOAcc case, the surfaceSpec can use any new variables. These

[9]ports and fields share the same name and index

variables can be appended to the end of the input argument list of this function, like the pair ['tiltX', 1e-6]. The value can be specified by a variable as in the FPIFOAcc example, then they can be changed by using runtime variables.

- **Example of serHRfiles with thermal distortions**

  ```
  obj.setHRfiles( 'ETMX', ETMXdat,
  'map % phasemap specified by ETMXdat data file
  + calcThermal(0,r,0,PcoatEX,0.062,0.2,0.17) %coating absorption
  + pointAbs_SINB(x,y,xPoX,yPoX,PpointEX)' %point absorbers
  0.16, 1/measuredRoC, 0,
  'PcoatEX', PcoatEX, %coating absorption setting 'xPoX', xPoX,
  'yPoX', yPoX, 'PpointEX', PpointEX ); %point absorbers setting
  ```

- **`setTRfiles('mirName', 'mapFile', 'transSpec', apertureRoC, measuredRoC, orientation, 'name1', val1, ...)`**
  - This defines the details of the transmission through the mirror 'mirName', from AR side to HR.
  - important sign convention The data measured by Fizeau IFO is the optical path distortion (OPD) and the value is positive when the optical path change is negative or smaller. The concave surface is $r^2/(2 \cdot RoC)$ with $RoC > 0$. So the focusing lens in a substrate is $r^2/(2 \cdot RoC)$ with $RoC > 0$.
  - 'transSpec' specifies the substrate transmission map by using expressions supported by matlab. The typical form is 'map + func(x,y,r)'. func is negative of the optical path length, to match with the convention of map. When calcThermal or pointAbs is used to implement the lens, make sure they are added with negative sign, i.e., **`transSpec = map - calcThermal(-1,...)`**. The -1 in the first argument means it is the substrate lens optical path without surface shape change. SIS adds the optical path by the surface distortion.

4.7 Field analysis

- **[E,x,y] = ifo.getField(fieldID)** returns the amplitude of the field. For **`fieldID = 'ITM-HR-o'`**, E is the amplitude of the field going out of the HR side of ITM. Because of the matlab (row, column), or horizontal-vertical convention, E(j,i) is the amplitude at (x,y) = (x(i), x(j)).
  `pcolor(x,y,abs(E).^2)` is the power distribution with the x in the horizontal direction and **`plot(x, abs(E(end/2,:)).^2)`** is the power distribution along the x axis at y=0.
- **`ifo.HGPower(fieldID, m, n)`**, **`ifo.LGPower(fieldID, l, m)`**
  gives the power of Hermite Gauss, Laguere Gauss field of the specified mode.
  When the first mode index, m for HG and l for LG, is negative, the function returns an array of mode powers. $nmmax = |m|$ or $|l|$ in the following case list.

- n = 0 : sum of power of modes with n+m=nmmax :
  HGPower(-2, 0) = HG(0,2) + HG(1,1) + HG(2,0);
  LGPower(-2, 0) = LG(1,0) + LG(0,2) + LG(0,-2);
- n = 1 : power of elements for $|2p + m|$:
  HGPower(-2,1) = [HG(0,2), HG(1,1), HG(2,0)];
  LGPower(-2,1) = [LG(1,0), LG(0,2), LG(0,-2)];
- n = -1 : array of powers with $m + n <= nmmax$:
  HGPower(-2,-1) = [HGPower(0,0), HGPower(-1,0), HGPower(-2,0), sum of powers with $n + m > 2$];
  LGPower(-2.-1) = [LGPower(0,0), LGPower(-1,0), LGPower(-2,0), sum of powers with $n + m > 2$];

• **[modeStr, modes] = ifo.mainHGsStr(fieldID, range1, range2, thershold)**, **[modeStr, modes] = ifo.mainLGsStr(fieldID, range1, range2, thershold)** returns a list of the HOMs of the field, sorted by power power.
Modes sorted are defined by range1 and range2, like range1 = 0:20, range2 = 0:20 for HG cases, and range1 = 0:10, range2 = -20:20 for LG case. These are default values when range1 and range2 are empty. modeStr is a readable text format of numerical array mode.

modes(index,1:2) = mode index (m,n, or p,m)
modes(index,3) = amplitude
modes(index,4) = power fraction
modes(index,5) = fractional loss of this mode
modes(index,6) = amplitude of this mode at the source
modes(index,7) = fractional loss of this field $(ind, 5) * (ind, 6)^2/fldPower$
threshold $\geqq 1$: returns modes whose fractions is among the top N = int(threshold)
threshold < 1 : returns modes whose fraction is larger than threshold
threshold $= 0$: all
threshold < 0 : max(ceil(length(range1) * length(range2) / 20), 10);

```
Example of fp.mainHGsStr('ETM-HR-i','','',5)

index]   (m, n)   fraction / power   loss (frac*loss)   amplitude
--------------------------------------------------------------------
1] ( 0,0) 9.9992e-1/2.6602e+2  6.6672e-7(6.6666e-7)  1.6310e+1+2.2190e-2i
2] ( 2,0) 1.8515e-5/4.9258e-3  2.1124e-3(3.9190e-8) -5.3361e-2+4.5590e-2i
3] ( 0,2) 1.6439e-5/4.3736e-3  2.2772e-3(3.7516e-8) -4.9803e-2+4.3511e-2i
4] ( 0,4) 1.1947e-5/3.1784e-3  4.3667e-3(5.2166e-8) -1.3126e-2+5.4828e-2i
5] ( 4,0) 1.0994e-5/2.9248e-3  4.3615e-3(4.7946e-8) -1.1535e-2+5.2837e-2i
-----
fraction not covered by these modes =   2.6629e-05
total of frac*loss =   8.4348e-07
```

4.8 Cavity analysis

- **RTL = ifo.roundtripLoss(fieldID)** calculates the round trip loss of the propagation staring from fieldID, and back to fieldID. If no fieldID is specified, round trip losses of all cavities are calculated. The order of the closed cavities can be found by ifo.cavities.printAll.

4.9 Analysis support

- **`[dE, E0, xl, dP, L] = ifo.getPointScatter( fieldID, defects, WfftOut, L )`**
  - This calculates the far field dE induced by point scatterers on the surface where fieldID comes out (e.g., if fieldID = 'ITM-HR-o', the surface is ITM) whose shapes and distributions are defined by the parameter **defects**. See getPointScatter.m for details.
  - The field induced by the point scatterer are calculated distance L from the source in a area width of WfftOut.
  - defects is an array of information of defects, like

```
 defects = {u0(1), v0(1), du(1), dv(1), rms(1);
            u0(2), v0(2), du(2), dv(2), rms(2);
            ...}
(u0, v0) : location of defect
(du, dv) : shape of defect
   1) du>0 : for square case, size of (2xdu, 2xdv)
   2) du=0 : for circle case, radius = dv
   3) du<0 : for ring case, 2xdu is the thick and dv is mean radius.
rms : the average height of the aberration
```

- **`[dat, coef, nmList] = zernikeCoef( dat, xl, yl, aperture, nmList[3], remove[1], normalize[false], w[0] )`**
  - This calculates the zernike coefficients of dat defined in [xl,yl], and returns a dat with specified zernike terms subtracted (if remove is true).
  - Zernike coefficients are calculated using data in aperture, with gaussian weight of w if it is specified.
  - zernike modes calculated are specified by nmList.
    nmList = 0 36 : first nmList+1 Wyko terms are handled (0:piston,1:tiltx,2:tilty,3:power,4:Astigm cos(2 theta), 5:Astigm sin(2 theta)). Default is nmList = 3, which removes up to power.
  - tilt is (n,m) = (1,1) and (1-1) and theta = coef / radius
    power term is (n,m) = (2,0), and $RoC = radius^2/(4 \cdot coeff)$
    for unnormalized $zernike(2,0) = 2 * rho^2 - 1$.

- **`[D, rList, phiList] = UtilTK.xy2r( xlist, xyDist, rList, x0, y0, N[1], dphi[0] )`**

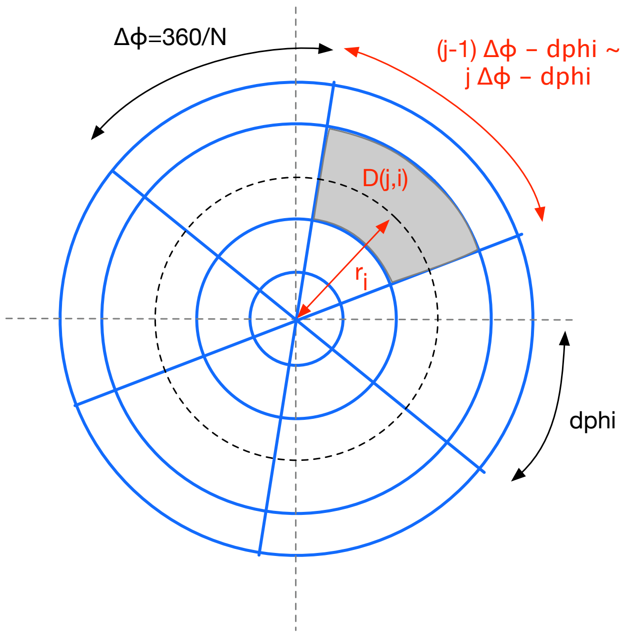


Figure 4: Segment of D(j,i)

  - Given a distribution specified by xyDist(j,i) at (x,y) = (xlist(i),xlist(j)), this function calculates distribution in the r-phi coordinate.
  - For a given list of rList = [r1, r2, ...,] and number of phi segment N, density D is calculated, defined as
    $S(j,i) = \iint\limits_{V} dxdy \;\; xyDist(x,y);, D = S/V;$
    where V is the area shown in gray in the figure, i.e., $r = (r(i-1)+r(i))/2 \sim (r(i1)+r(i+1))/2$ and $phi = (j-1)(360/N) - dphi \sim j(360/N) - dphi$.
  - When the last radial value of rList is 0 (rList(end)==0), P is returned.
  - rList in the return value is the same as the input, except (1) the first one in the input may be different when r1 = 0, and the last one is removed if rList(end) = 0.
  - phiList(j) in the return value is $360/N \cdot j - dphi$.

4.10 Optics surface

- **`[map, xList] = ifo.optics.getHRmap(optID, removeRoC[true], Aperture[])`**
  When the surface map is defined in setHRfiles, e.g., by additing thermal deformations, this function can be used to see the HR surface map used to calculate fields.
  `removeRoC` removes the spherical curvature shape, and the map is filled by nan out of **`Aperture`** if it is specified.

4.11 Modal Model Toolkit - static member functions

- **[z, z0] = MMTK.FPparam(invRoC1, invRoC2, cavLeng)**
  returns the z of the front mirror and the Rayleigh range of a stable FP cavity. invRoC1 anc invRoC2 are inverse of the radius of curvatures of the front and back mirror surfaces facing each other, separated by cavLeng. z ¿ 0 when invRoC1 ¿ 0.
- **[w1,w2] = MMTK.w1FP(invRoC1, invRoC2, cavLeng, waveNumber [**$2\pi/1.064m$**])**
  returns the beam size at the input and end mirrors of a cavity. When the waveNumber is ommited, the default value for $1.064\mu m$ is used.
- **eta = MMTK.gouy00FP(invRoC1, invRoC2, cavLeng)**
  returns the gouy phase of a FP cavity.

5 Objects in SIS

The base class of the SIS simulation package is FFTIFO. This provides the framework to define the optical configuration using the member function defineIFO by adding optics (addMirrors) and specifying connectors (addProp). The latest SIS has a derived class FFTIFOAcc, which includes the acceleration scheme by C.Day and G.Vajente (find out more here). It also provides various analysis tools.

To start the SIS project, the user has to create a derived class of **FFTIFO**, and specify the member function **FFTIFO.defineIFO**. If the mode base is to be specified explicitly, **FFTIFO.setupBaseMode** is to be specified. So the IFO definition file will look like the following.

```
classdef myIFO < FFTIFOAcc  % it is derived from FFTIFOAcc
    methods
        function obj = myIFO( varargin )
            obj@FFTIFOAcc( varargin{:} );
        end
        function defineIFO( obj, varargin )
            % you codes defining IFO
        end
        function setupBaseMode( obj, varargin )
            % your code defining the mode base
        end
    end
end
```

This is the object structure of a FP cavity, but it is the same for any IFO.

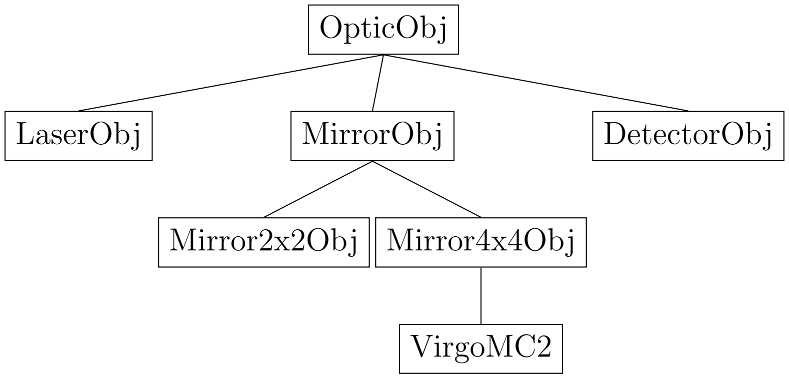
```
>> fp = FPIFOwMaps;
>> fp
```

```
fp =

  FPIFOwMaps with properties:

           ffts: [1x1 FFTTK]
          twids: [1x1 twiddle]
         optics: [1x1 opticManager]
          props: [1x1 propagaterManager]
         fields: [1x1 fieldManager]
       cavities: [1x1 cavityManager]
          debug: [1x1 DebugTK]
    savedFields: [0x0 struct]

>> fp.fields

ans =

  fieldManager with properties:

          ifo: [1x1 FPIFOwMaps]
    fieldList: [1x11 FieldObj]
        lamCR: 1.064000000000000e-06
          kCR: 5.905249348852994e+06
          kSB: 0

>> fp.cavities

ans =

  cavityManager with properties:

                ifo: [1x1 FPIFOwMaps]
               tree: {[2 6 -1]  [2 5 9 5]  [2 5 9 6 -1]  [2 5 10 -11]}
           fullTree: {[2 6 -1]  [2 5 9 5]  [2 5 9 6 -1]  [2 5 10 -11]}
         closedCavs: {[5 9 5]}
    lengthFreezeFlag: 0
           compList: [1x1 CompObj]
          guessList: [1x2 GuessObj]
           pathBook: {2x5 cell}
```

At the beginning of the construction of FFTIFO, various empty array of objects and the manager of these objects are created. Some important ones are:

- optics
  - opticManager

  - array of optics of various classes derived from opticObj



- props
  - propagatorManager
  - array of propObj
- fields
  - fieldManager
  - array of fieldObj
- cavities
  - cavityManager
  - various kinds of properties

Manager objects make it easy to access and manage elements of the objects in their arrays.

All objects in arrays have names, either specified by the user or named automatically. **fieldManager.findNameField('fieldName')** returns the serial index of the field specified by the name. **opticManager.findLaser** returns the index of the laser object. All managers have the printAll function, which prints information of the objects in the group.

When addLaser and addMirror are called, a new opticObj is created and it is added to the array. When one optic element is created, fields are created for all the ports of the optic. E.g., Mirror2x2, like ITM, has two input ports, on HR and AR sides, and two output ports. Also, all ports have associated fields created. This process expands the array of fieldObj.

When addProp is called, one or two props are added to the propObj array in props. addProp calls one or two addOneProp, depending on the links. When the propagation defined by the addProp can propagate backwards, like addProp('ITM-HR-o', 'ETM-HR-i'), it calls addOneProp('ITM-HR-o', 'ETM-HR-i') and addOneProp('ETM-HR-o', 'ITM-HR-i'). If the propagation in addProp does not have a backward pass, like addProp('IMC1-HR-o', 'IMC3-HR-i'), only one propObj is added.

When the IFO configuration is analyzed, many elements in cavities are added and updated so that the field propagation paths can be recognized and traced for the calculation of the field evolution. Once analyzed, most of the planar optical configurations can be recognized and the fields can be calculated (not completely true, but almost).

6 Simple flow of SIS20

In order to construct the IFO within SIS, the function FFTIFO plays the main role.

First of all, it prepares "everything" for the simulation, naming the object by: obj. Then, it starts calling a series of function that we will later on describe.

```
obj.ffts = FFTTK;
obj.twids = twiddle( obj );
obj.fields = fieldManager( obj );
obj.optics = opticManager( obj );
obj.props = propagaterManager( obj );
obj.cavities = cavityManager( obj );
obj.debug = DebugTK( obj );
```

In what follows, we describe each of these lines of code

1. In this line we invoke the basic framework for the FFT calculation
2. **Twiddle** is a field calculation code that uses the plane wave approximation. It is used to prepare the initial condition in SIS, since the convergence of the field calculation is faster if it starts from a good initial condition, i.e., close to the final answer [2].
3. **fieldManager** is a function that calls **FieldObj**. The latter is a structure that contains all the information regarding the field:
   - **invRoC**: inverse radius of curvature[10]
   - w: beam waist
   - z_0,z: initial and final points of the path followed by the field
   - ϕ_{Corr}: change of phase due to the propagation or, said otherwise, the distance traveled by the field (except for a factor $\frac{4\pi}{\lambda}$)
   - **EFFT**: value of the field (in internal format) in each grid point
   - status: flag that indicates which fields are left to calculate (default value: 0)
   - **portIDfrom**: it informs on whether the field comes out from a port in a non-standard way (default value: 0)
4. **opticManager** is a function that calls **OpticObj**. The later is a structure where the in and out ports from a certain optic are created. Also, it assigns a unique ID to this optic. It includes the following information
   - **ApertureDef**: when its value is < 0, the aperture is calculated to be the beam size times the absolute value of ApertureDef [-8 by default]

[10]More convenient to use than the radius of curvature itself, given that we normally use very large values for the later, such as ∞ (i.e.:flat mirrors)

- **virtualAperture**: defines extra outer space in which the field is to be calculated [0 by default]
- **WfftUser**: is a variable that makes the Wfft (the size of the FFT window) the same on the test masses. Also, it can make the Wfft be large on RMs without making other Wfft larger. Summarizing, the way it works is as follows:

$$WfftUser = \begin{cases} > 0 & Wfft = WfftUser \\ < 0 & Wfft = wOpt \cdot |WfftUser| \end{cases}$$

  where *wOpt* is the typical beam size. The number of points for the FFT (Nfft) remains constant. WfftUser has a default value of 0.
- **optX0, optY0**: are the coordinates of the mirror centre [(0,0) by default]
- **invRoCAR, invRoCAR**: inverse radius of curvature of the HR and AR sides, respectively, of an optic [both are 0 as default]
- **inPorts, outPorts**: matrices representing the in and out ports of the optic
- **reflMatch, transMatch**: these variables define the direction of the reflection and transmission
- **reflABCD, transABCD**: contain the analytic values of the ABDC matrix. These values are not used for the FFT propagation, but it helps to set up various values
- **resolution**: its minimum value is given by the beam size divided by 16 (value that allows to recognise the Gaussian structure). Alongside with the value of the resolution we must consider the adaptive grid size in all optics, and depending on which method sets a more stringent requirement, that will be the value to be used as resolution [-16 by default].
- **backScatter**: backscattering on the HR side [false by default]

5. propagaterManager
6. cavityManager
7. DebugTK

The next lines of code within FFTIFO are the following:

```
obj.defineIFO( varargin{:} );
obj.analyzeCavity;
obj.setupBaseMode( varargin{:} );
obj.updateNfft( varargin{:} );
obj.setupMisc;
obj.startFFT;
obj.twiddler;
obj.calcIt( varargin{:} );
obj.clearSavedFields;
```

We now describe these lines of code

1. **defineIFO** fills all the arrays representing optics and propagators (this is a mandatory method to be prepared)
2. **analyzeCavity** it finds the connections between optics, e.g.: it defines where the laser comes in from, whether it is transmitted, reflected, ...
3. **setupBaseMode** initializes the parameters to start calculating all modes everywhere, assuming a perfect scenario where there is no aberration, optics have infinite curvature, ... From the procedure we end up obtaining the beam size and its radius of curvature
4. **updateNfft** is used to define the number of grid points and the window size. The later has to be about twice the optic size so as to avoid aliasing (check
5. **setupMisc**: this function is in charge of setting up surface maps information (i.e.: including mirror maps).
6. **startFFT**: this function fills all EFFT by empty field. This is done in order to make EFFT be a valid matrix for those ports which do not have source, or status = -1
7. **twiddler**: calculates the power using the scalar field analytic formula
8. **calcIt**: is a function that allows to calculate whatever the user desires
9. **clearSavedFields**: is in charge of saving and restoring fields so as to restore a given state

A Acronyms

SIS: Stationary Interferometer Simulation

AR: antireflective

HR: highreflective

SLS: stray light simulations

IFO/ITF: interferometer

(I)FFT: (Inverse) Fast Fourier Transform

IMC: Input Mode Cleaner

PDH: Pound Drever-Hall

RMS: Root Mean Squared

HOM: Higher Order Mode

PD: photodiode

(inv)RoC: (inverse) radius of curvature

FSR: free spectral range

PA: point absorber

TD: thermal deformation

FP: Fabry-Pèrot

ITM: Input test mass

ETM: End test mass

TCS: Thermal Compensation system

LG/HG: Laguerre Gauss/Hermite Gauss

RF: radio-frequency

B Caveats

There are several simple caveats to know to have correct answers. This section summarizes those points.

B.1 Clipping by FFT window, or power from ITM is not the same as the power to ETM

When the field propagates, the beam spread becomes larger. Even when the gaussian beam power distribution becomes narrower by the waist position, the tail of the field spreads caused by the abberation of the mirror surface, i.e., surface structure with small spatial wavelength. When the field propagation from ITM to ETM is calculated, the field leaving ITM is limited in the aperture of ITM, which is inside of the FFT window where the field is calculated. When the field propagates towards the ETM, the field shape changes and the field on ETM could spreads to outside of the FFT window as is shown in Fig.5.

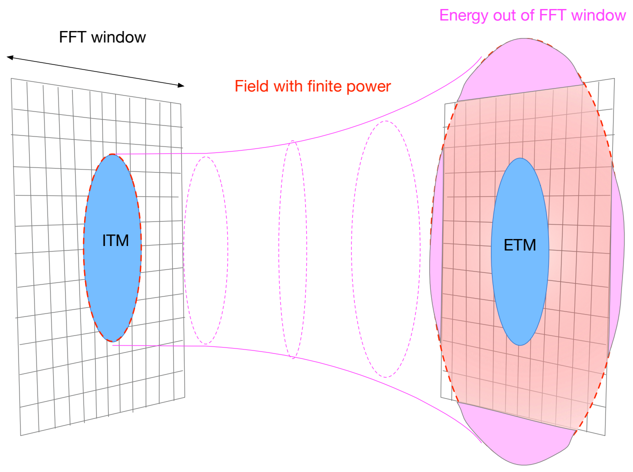


Figure 5: Field clipping by FFT window

When powers are calculated, SIS sums up all elements in the FFT window. As to the field going out of ITM **`'ITM-HR-o'`**, the field with finite amplitude is in the FFT window, and the total power is calculated properly by **`PfromI = power('ITM-HR-o')`**. But, on ETM, part of the field with finite amplitude could be out of the FFT window (pink area in Fig.5), and the power calculated using the sum of all FFT window area may not be the total power of the field on ETM.

The field amplitude in the ETM plane is correct in the aperture of ETM, and the power hitting ETM can be calculated by **`PtoE = powerInR( 'ETM-HR-i', radiusOfETM )`**. To calculate the clipping loss due to the ETM geometry is to be calculated by **`PfromI - PtoE`**.

Usually, the FFT window size is chosen so that this kind of clipping loss is negligibly small. But, when the loss balance is calculated with high precision, this loss may not be negligible. When the total power is needed, make sure to use the field which is known to be in finite area, like the field going out of a mirror.

C printAll output example

This is an excerpt from the output of fp.printAll('FPIFOAcc.txt') with some comments. Comments are shown in blue. Full text is Codes/Note1/FPIFOAcc.txt.

```
 === Information about fields ===

      Field going to ETM.
      Power is the result of FFT simulation
      all other values are calculated using the design parameters
7) name: ETM-HR-i   ====   Power = 283.434, E(analytic) = 16.8378, [status = 1024]
RoC = 2245, w = 0.0619634, z = 2160.28, z0 = 427.807, 1/phiCorr = 2153.05
q = 2160.28 + i 427.807, 1/q = 0.000445434 - i 8.82107e-05

 === Information about optics ===

      ETM optic information
3) name: 'ETM',  Wfft = 0.991414,  Nfft = 256
      loss of each port is the power going to the port (ETM-HR-i)
      minus sum of powers of going out (ETM-HR-o + ETM-AR-o)
      input power is calculated using the source, ITM-HR-o, not ETM-HR-i,
      because the ETM-HR-i can be calculated only in limited area
      port numbers [7, 8] = [ETM-HR-i, ETM-AR-i] and [9, 10] = [ETM-HR-o, ETM-AR-o]
inPorts  = [ 7 8 ]  loss for each input port = [ 1.25944e-06 NaN ]
outPorts = [ 9 10 ]

      For the calculation of the total loss of this optic,
      input powers are calculated using the source powers as mentioned above.
loss 0.000332876 in 'ETM-HR-i'+'ETM-AR-i'->'ETM-HR-o'+'ETM-AR-o'

      optical parameters of ETM
R = 0.999995, T = 5e-06, L = 0
Roc(HR) = 2245, Aperture = 0.326000, Thickness = 0.000000, n = 1.449630,
Inc.Angle = 0 (degree), theta = 0, phi = 0
wOpt = 0.0619634, Wfft = 0.991414
      ABCD matrix based on design values of refractive index, RoC and thickness.
ABCD matrix
   refl  from 'ETM-HR-i'[ 7] to 'ETM-HR-o'[ 9] = [ 1,  0; -0.00089087   1]
   refl  from 'ETM-AR-i'[ 8] to 'ETM-AR-o'[10] = [ 1,  0;  0.0012914    1]
   trans from 'ETM-HR-i'[ 7] to 'ETM-AR-o'[10] = [ 1,  0; 0.00020028    1]
   trans from 'ETM-AR-i'[ 8] to 'ETM-HR-o'[ 9] = [ 1,  0; 0.00020028    1]

      specification of the surface structure set by setHRfiles
File lists in 'HRfiles'
  [1] no file specified :      formula = 'map + tiltX*x'
      aperture = 0.16 :  RoC = 2245 :
      golden rule : loss1 (lam > 3.87271 (mm)) = 0 (ppm), loss2 (lam > 3.87271 (mm)) = 0 (ppm)
  [*] values of 1 variables
```

```
      (1) tiltX = 0

 === Information about propagators ===

      All quantities are based on design parameters specified by addMirror and addProp
      prop loss is the source power - the power on the end mirror
3) name: FPprop   prop loss = 1.28523e-06
Prop from 'ITM-HR-o'[5] to 'ETM-HR-i'[7]
L0 = 3994.5, propL0 = 3994.5, delL = 4.6009e-07, lockPhase/pi = 0, gouy00 = 2.71695
nRef = 1, ratio = -1.16921, CLa = -0.000464458, (-k*L+(1+n+m)gouy00)/pi = 4.65669e-08
propABCD = [         1,     3994.5;          0          1]

      When FPprop is added, this back scattering is automatically added.
4) name: FPprop_Bck   prop loss = 6.43953e-07

 ==== Closed cavities
[ 1]  ITM-HR-o (5)  ->  ETM-HR-o (9)  ->  ITM-HR-o (5)
Eigen value at port [5] : RoC = -1934, w = 0.0529939, q = -1834.22 + i*427.807
Round trip loss is 1.9033e-06 using guess ID 1
Large angle scattering loss due to downsampling from 0.003873 to 0.003873 is 0
```

References

[1] J.-Y. V. Patrice Hello, “Analytical models of thermal aberrations in massive mirrors heated by high power laser beams,” Journal de Physique, vol. 51, pp. 1267–1282, 1990.

[2] J. E. M. Martin W. Regehr, “Twiddle. a program for analyzing interferometer frequency response,” LIGO-T960079-01-R, 1999.