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

# Twiddle (ver. 3.0)

**A Program for Analyzing Interferometer Frequency Response (Mathematica 3.0)**

Martin W. Regehr

James E. Mason

Hiro Yamamoto

**LIGO-T990022-00-R**

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

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

# 1 INTRODUCTION

In the design of an interferometer length control system, it is necessary to know the transfer functions of the interferometer (which is sometimes called the "plant" in this context). Typically the inputs of this plant are the positions of the mirrors and the laser frequency. The plant outputs are often demodulated photocurrents, from photodiodes which measure the intensity at the optical outputs of the interferometer. A transfer function from one of the inputs to one of the outputs is then defined in the usual way: it is a function of frequency such that its amplitude and phase give the ratio of signal at the output to signal applied to the input.

The *twiddle* program was written to analyze the transfer functions between motion of optical components and output voltages, in interferometers illuminated with light which is in general made up of a carrier and one or more sets of sidebands[1]. The output voltages are assumed to be derived by sinusoidal demodulation at specified frequencies of the light intensity at the optical outputs of the interferometer, and the motions of the optical components are assumed to occur at frequencies small compared to the modulation frequency. It is assumed throughout that only a single transverse mode of the light is present. The program analyzes only motion along the beam axis.

Twiddle consists of four Mathematica packages: *twiddle.m* and four subroutines called by Twiddle: *operations.m*, *components.m*, and an *ifo.m* notebook. The *ifo* notebook is to be built by the user, whereas the former two are transparent to the user. *ifo* contains a list of commands interpreted by *twiddle* as specifying the types and locations of optical components in the interferometer.

During a normal session with the program, the user will first construct or modify a notebook containing the specifications of the interferometer. This is the *ifo* notebook. The specifications consist of information about the reflectivity, transmission, and the relative spacing of the optical components, as well as information regarding the conditioning of the input light. In another working notebook, *twiddle* is run, which sets up the static interferometer. At this point, any errors in the setup of the interferometer are reported. Next the user will specify information such as frequency range, resolution, which mirrors to shake, and at which places in the interferometer the transfer function is to be measured. Finally the transfer functions are generated, and either displayed as plots or written to files.

Chapter 2 is a tutorial with two examples. A Fabry-Perot cavity is analyzed with the purpose of introducing *twiddle* and the various parts of *twiddle* with which the user interacts. Commands used and the form of the user specified notebook *ifo* is demonstrated. A second example, a Michelson interferometer, is approached by starting with a sketch of the interferometer and working through the necessary steps to build the *ifo* notebook. Chapter 3 describes the general theory which *twiddle* uses to calculate the transfer functions. Chapter 4 gives a list of commands available to the user along with syntax and description.

Please report any problems or bugs to:

jim@ligo.caltech.edu or hiro@ligo.caltech.edu

Thank you!

1. Specifically, Twiddle has two modes for the input light. It either assumes that the input light is phase modulated at a single frequency to the order specified by the user, or is comprised of an arbitrary set of sidebands, of which the frequency and amplitude are specified by the user.

# 2 CHAPTER 2 TUTORIAL

In this chapter we will set up and analyze two simple interferometers: Fabry-Perot, and Michelson, using the example notebooks that have been supplied.

## 2.1 The Fabry-Perot Cavity

The first example is a Fabry-Perot cavity. We will look at the two methods for specifying the input light in *twiddle*, the first assuming a single phase modulated frequency, and the second where the user can specify arbitrary frequencies and arbitrary amplitudes for the sidebands.

### 2.1.1 FP cavity (single frequency)

We will model a Fabry-Perot cavity on resonance, using light which is phase modulated at a single frequency, to first order. The carrier is resonant, and the sidebands are exactly anti-resonant. Implicit in this assumption is that the second order sidebands are negligible (weak modulation limit). The *ifo* notebook is *fp_ifo.nb*. Open a working notebook in Mathematica, as well as the *fp_ifo* notebook. This is what the *ifo* notebook looks like:

```
(* fp_ifo   :   program to set up Fabry-Perot cavity. *)

loss = 100. 10^-6
cT1 = .03
cT2 = loss
other[cT_] = 1. - loss - cT
cR1 = other[cT1]
cR2 = other[cT2]
r1 = Sqrt[cR1]
r2 = Sqrt[cR2]
t1 = Sqrt[cT1]
t2 = Sqrt[cT2]

gamma = 0.1

props = {{ 1. ,-1.}},
         { 1. , I },
         { 1. , 1.}}

s1 = source[gamma]
m1 = mirror[r1,t1]
connect[s1,1,m1,1,0.]
m2 = endmirror[r22,t2]
connect[m1,2,m2,1,3987.2397, 12.5 10^6]
```

It is imperative that the commands in this file be specified in **initialization** cells, so that when the notebook is saved, it creates the file *fp_ifo.m*, which is the file that *twiddle* looks for when executed.

Let’s first look at what this file does, before we calculate a transfer function. The first 11 lines of the program provide the necessary specifications of the optical elements in the Fabry-Perot. In general, the user needs to specify the amplitude reflectivity and transmissivity of each element. Here, the transmission coefficients are given as cT1 for the input mirror and cT2 for the end mirror, and, based on the assumption of 100 ppm losses in each optical element, the necessary parameters are calculated for the user, as r1, r2, t1, and t2.

The line

```
gamma = 0.1 (* modulation depth *)
```

specifies the depth of the phase modulation of the light incident on the interferometer.

The next definition is very important.

```
props = {{ 1. , -1.},
        { 1. , I  },
        { 1. , 1. }}
```

is a matrix which specifies the exponential of the one-way propagation phase, $e^{-i2\pi\frac{(length)(frequency)}{c}}$, modulo 2π, of each of the three RF frequencies (the carrier and an upper and a lower sideband) as they travel from one optical component to the next. The phases are absolute, that is, the sideband frequencies are the carrier frequency ± the modulation frequency. `props` must contain one row for the each RF frequency and one column for each path between two optical elements. The rows are organized from lowest sideband in the first row to the highest sideband in the last row. The order of the columns is dictated by the order in which the optical elements are connected, which will become clear later. In this example, we need to connect the source to the input mirror, and the input mirror to the end mirror. We see that all three frequencies arrive in phase at the input mirror, which is represented in the first column. The second column shows that the carrier gets 3π/2 phase (Exp[-I 3π/2] = I) from the input to the end mirror, and so a round trip accumulates 3π. Note that there must be a phase reversal upon reflection from one of the mirrors to increase this to 4π, so that the carrier will resonate. The sign convention for the amplitudes of reflectivity is discussed later. We also see that, in this particular case, the sidebands are leading and lagging the carrier by π/2, so they will be exactly anti-resonant in the cavity. Later, we will specify the length of the cavity, and the modulation frequency. *twiddle* has a consistency check which will then calculate the relative phase of the sidebands with respect to the carrier, to ensure these phases in `props` are consistent with the lengths and frequencies specified. The default for the number of RF sidebands considered is two (one on each side of the carrier). This can be changed by typing `sborder = `*n* inside Mathematica before running *twiddle* (to cause *twiddle* to consider n RF sidebands on each side of the carrier) and adding rows above and below the existing rows of the matrix `props`.

Next, we must define all optical components, and specify the relations between them. The definition of components is accomplished by assigning a variable name to each component, using the appropriate command defined in *twiddle*. Four component commands are available, `source`, `mirror`, `endmirror`, and `beamsplitter`.

```
s1 = source[gamma]
```

defines a source of phase modulated light, modulated at modulation depth `gamma`, named s1.

```
m1 = mirror[r1,t1]
```

defines a mirror, named m1, of amplitude reflectivity r1 and transmissivity t1. A mirror is defined to have two "ports", where a "port" corresponds to the existence of an optical path to the component. Since the input mirror has light incident from both sides, it has two ports. We will see in the next example that a beamsplitter (simply a mirror at non-normal incidence) has four ports. Corresponding to each port are two directions defined for incident and reflected light. On page 11 are shown the various optical components and the sign and numbering conventions which are used by *twiddle*. The sign convention is that of the Stokes relations for reflection and transmission. It is imperative that these conventions are recognized. For this example, we see that the program considers the reflectivity of the mirror to be positive at port 1 and negative at port 2, as we had deduced earlier by demanding the carrier be on resonance.

As components are defined, you can specify the connection between them. It is important to note that the connections must be made consistent with the column ordering of the `props` matrix.

```
connect[s1,1,m1,1,0.]
```

connects port 1 of the source s1 to port 1 of the mirror m1, with an intervening space of 0 m. The first column of props corresponds to this connection.

```
m2 = endmirror[r2,t2]
```

defines the second mirror.

```
connect[ml,2,m2,1.3987.2397, 12.5 10^6]
```

connects port 2 of mirror m1 to port 1 of mirror m2, with an intervening space of 3987.2397 m. (Twiddle uses the value c = 2.99792458e+8). The second column of props corresponds to this connection. The final argument of this last `connect` command is optional; it gives the modulation frequency in Hz. If this argument is included in a `connect` command, the program will issue a warning if the phases indicated in `props` corresponding to the sidebands differ by more than $1°$ from the values calculated using the carrier element, the length of the connection, and the frequency specified. This is the consistency check mentioned earlier.

We’re now ready to model this interferometer. In order to model this case, we will work in the new notebook. In the first evaluation cell, we input

```
SetDirectory[“/home/jim/twiddle/twiddle_2.2/test”];
IFOName = “fp”;
sborder = 1;
<<“twiddle.m”
```

The first `SetDirectory` command tells Mathematica where the files it needs reside. This is useful if you don’t start up Mathematica in the working directory. The second command, `IFOName = “fp”;` tells Twiddle what file contains the interferometer configuration; if `IFOName` is defined like this, Twiddle looks for *fp_ifo.m* (Twiddle appends the “ *_ifo.m*”), if no `IFOName` is defined, Twiddle looks for *ifo.m*. The `sborder = 1;` command tells Twiddle to what order in the series expansion of the phase modulation to go. The default is 1, so the definition is redundant here, however to demonstrate that this is where such a command would go, it’s included in this example. Finally, `<<“twiddle.m”` is run, which sets up the static fields for each frequency of input light everywhere in the interferometer.

When this is run, Mathematica outputs

```
Twiddle version 3.0, February 1999
Martin Regehr, James Mason, Hiro Yamamoto
California Institute of Technology
Processing fp_ifo.m
Input amplitudes = {-0.0499375 I,0.997502,-0.0499375 I}
Finished processing fp_ifo.m
Matrix size is 9
```

This output basically indicates that the *fp_ifo.m* interferometer was successfully built. If there were any mistakes in how the optics and optical paths were setup, various error messages would be displayed here. Once this interferometer setup is complete, it does not have to be repeated unless changes are made to the interferometer itself.

Let’s Bode plot! In the next evaluation cell, input

```
ClearBuff;
npoints = 20;
startfreq = 1;
stopfreq = 10000;
shake[m2];
outindex = index[s1,1,1];
demod = 1;
CalcAndPlot[demod,outindex]
```

`ClearBuff;` is just a good idea. This command should preface every transfer function run. The next three lines

```
npoints = 20;
startfreq = 1;
stopfreq = 10000;
```

defines the frequency span that will be analyzed, and the number of points, logarithmically spaced, that will be used. These variables need to be defined globally.

```
shake[m2];
```

tells Twiddle which component to shake. It’s possible to shake more than one component at a time, as we’ll see in the next section. Here, we’re shaking the 2nd mirror we defined in *fp_ifo*, m2.

```
outindex = index[s1,1,1];
```

tells Twiddle where, in the optical configuration, to demodulate the light. This is almost like placing a pickoff where we wish to find the transfer function. The arguments of the function index refer to the component name, the port, and the direction, in that order. In this case we are demodulating the light returning to the source s1 at port 1, from direction 1 (see page 15 for the definition for sources and mirrors and the like).

```
demod = 1;
CalcAndPlot[demod,outindex]
```

generates the Bode plot. `demod = 1;` defines what demodulation to use; 1 indicates an inphase demodulation, 2 indicates a quadrature demodulation. `CalcAndPlot[demod,outindex]` generates the Bode plot. Note that `outindex` and `demod` are not necessary global variables. The last three lines could be replaced with

```
CalcAndPlot[1,index[s1,1,1]]
```

Evaluating this cell, we see progress indicated by the number of points (frequencies) that have been evaluated, then the magnitude and phase Bode plots are displayed. The frequency units are logarithmic. The units of the magnitude are in $\mathrm{volts}/(\lambda/(2\pi))$, assuming a photodiode of unity quantum efficiency and a transimpedance of 1 volt/amp (so it could be said that the units are really $\mathrm{Watts}/(\lambda/(2\pi))$). The magnitude is also scaled to an input power of 1 Watt. Furthermore, the transfer function is calculated using the fields specified at `outindex`, which frequently are the fields inside the interferometer, not picked off by some low reflectivity pickoff. In that case, the transfer function needs to be scaled by the power reflectivity of the pickoff, as well. This could be remedied by explicitly placing a pickoff in the optical configuration specified by the *ifo* notebook (It would most likely be modeled as a `beamsplitter` with very low reflectivity).

Now let's plot the response of the cavity to high frequency end mirror motions. Modify the frequency lines in the analysis cell to read

```
npoints = 100
startfreq = 100
stopfreq = 100000
```

and either re-run this evaluation cell or setup a new cell(You don't need to re-run the *twiddle* setup!). You should see a pair of Bode plots with resonances at multiples of the free spectral range frequency (37.5 kHz). Some of the features on these plots are too sharp to be resolved at the chosen resolution. If you're patient, you can increase `npoints` and re-plot; otherwise you can zoom in on selected features by changing the frequency bounds. What do you think is the cause of the notches between adjacent peaks? In the limit of very small mirror motions, shaking a mirror is equivalent to putting sidebands on the existing frequencies of light at the shaking frequency. We then can note that the notches occur at half free spectral range frequencies. These will be exactly anti-resonant in the cavity, and will not propagate to the output. It is important that the output of this program be believable.

#### 2.1.1.1 More tricks

Let's say you wish to look at the transfer functions at 2 ports. Simply change the second argument of CalcAndPlot to a list of indexes, and re-evaluate that cell.

```
outindex = {index[s1,1,1],index[m1,2,2]}
CalcAndPlot[1,outindex]
```

You will now see 2 Bode plots, distinguished by the text over them,

```
Output at index 1, demodulation 1, at frequency 1
```

and

```
Output at index 6, demodulation 1, at frequency 1
```

The index tells you which field number you're looking at, in the order specified. Here, "1" corresponds to the source s1, at port 1, in direction 1, while "6" corresponds to the field at m1, port 2, direction 2. The demodulation number is obvious, "1" for inphase, "2" for quadrature. We'll get to the frequency number later.

What if you want to shake more than one mirror at a time? Simply specify that by using more than one `shake` command. Here, we could do the following, replacing the single shake command with:

```
shake[m2];
shake[m1,-1];
```

The second argument in shake is optional, it indicates the magnitude and phase of the shaking (default is 1). In this case, mirror m1 is being shaken with the same amplitude, but opposite phase, as mirror m2.

Getting tired of trying to get the sideband phases in `props` right? If you know your modulation frequencies and optical lengths, fill the sideband rows with 0's, and feed the lengths and frequencies to the `connect` commands, and *twiddle* will fill the sideband phases for you.

Want to do the analysis using higher order sidebands? Simply change the definition of `sborder` to whatever you want, and make sure to add more sideband rows to `props`. Of course, for this you will have to re-run the *twiddle* setup. Also remember to save the *ifo* notebook so it generates an updated initialization file, first.

Want to output the results to a file? Replace the command `CalcAndPlot[demod,outindex]` with `CalcAndWrite[outindex]`. A file will be written in the working directory, using the assigned IFOName as a suffix to “.dat”, in our case, fp.dat. The data will be in columns, and both demodulations will be included. The first column will be frequency, the second column is the magnitude of the inphase demodulation of the first frequency at the first index, followed by the phase, quadrature magnitude, and quadrature phase in the third, fourth, and fifth columns, respectively. The following four columns are demodulated outputs at the second index. The organization of the output is that it loops over all output indexes at a given frequency of demodulation, then loops again over all output indexes at the next demodulation frequency. Multiple demodulation frequencies are discussed in the next section.

### 2.1.2 FP cavity (multiple frequencies)

The extension of this model to an arbitrary set of modulation frequencies and amplitudes is really quite simple. We’ll demonstrate this by using a somewhat artificial example: we’ll explicitly list the single phase modulated frequencies to second order. Let’s open up the notebook *fp2_ifo.nb.*

```
other[val_] := (1 - loss - val);
loss = 100. 10^-6;
cT1 =  0.03;
cT2 = 10 10^-6;
cR1=other[cT1];
cR2=other[cT2];
r1=Sqrt[cR1];
r2=Sqrt[cR2];
t1=Sqrt[cT1];
t2= Sqrt[cT2];

rf1 = 23.97 10^6;
rf2 = 2 rf1;

dgamma = {-BesselJ[2,.1],I BesselJ[1,.1],BesselJ[0,.1],
                I BesselJ[1,.1],-BesselJ[2,.1]};

modfreq = {-rf2,-rf1,0,rf1,rf2};

props = Transpose[{{1.,1.,1.,1.,1.},
                   {0 ,0, I, 0, 0 }}];

s1 = source[dgamma];
m1 = mirror[r1,t1];
m2 = endmirror[r2,t2];
connect[s1,1,m1,0.];
connect[m1,2,m2,1,3987.,modfreq];
```

The first 11 lines are the same as before - simply defining mirror amplitude reflectivities and transmissivities. The next two lines define the two sideband frequencies, rf1 and rf2. In our case, rf2 is simply 2 rf1, being the second order sideband. In general, of course, they can be any frequency the user desires.

Next is the first important difference.

```
dgamma = {-BesselJ[2,.1],I BesselJ[1,.1],BesselJ[0,.1],
                I BesselJ[1,.1],-BesselJ[2,.1]};
```

explicitly defines the amplitudes of each of the sidebands and carrier. In the single frequency case, the amplitudes are calculated by *twiddle* using the series expansion of phase modulation, given a modulation depth. In the multiple frequency case, each complex amplitude needs to be specified independently. This is useful if there are multiple modulations, either phase or amplitude, on the input light. The assumption is that there are as many frequencies above the carrier as below the carrier, however the frequencies above the carrier do not need to be the same as the frequencies below the carrier. Another important feature to point out is that *twiddle* will normalize the amplitudes so that the sum of the squares (or the input power) totals to 1. This is to preserve the notion that the transfer functions generated are normalized for 1 Watt of input power. A list of the frequencies used are given next, by

```
modfreq = {-rf2,-rf1,0,rf1,rf2};
```

It’s worth noting that the carrier is indicated by the “0”.

The rest of the notebook is pretty much the same, with the exception that the `props` matrix is taking advantage of the option to let *twiddle* calculate the sideband phases. In these cases, it’s never a bad idea to see what *twiddle* calculates, just to make sure it makes sense. Note also that the list of amplitudes becomes the argument for the `source` assignment. *twiddle* makes the distinction by determining if the argument of `source` is a number (modulation index) or a vector (list of amplitudes), and handles each case accordingly. Also, since we took the easy way out in building the `props` matrix, the list of frequencies needs to be passed into the second `connect` statement, otherwise it cannot calculate the proper phases. Furthermore, when `modfreq` is passed into `connect`, a global variable is created which is necessary when the demodulation process occurs. If `props` is built manually and `modfreq` is never passed to `connect`, a warning will be issued to assign this global variable manually.

Next, in the working notebook, *twiddle* is run and the interferometer is set up. This is done in the same way as before.

```
SetDirectory[“/home/jim/twiddle/twiddle_2.2/test”];
IFOName = “fp2”;
sborder = 2;
<<“twiddle.m”
```

The only significant difference is that, now, `sborder` indicates the number of frequencies on either side of the carrier. Running this, Mathematica outputs

```
Twiddle version 3.0, February 1999
Martin Regehr, James Mason, Hiro Yamamoto
California Institute of Technology
Processing fp2_ifo.m
Input amplitudes = {-0.00124896,-0.0499375 I,0.997502,-0.0499375 I,-
0.00123896}
Finished processing fp2_ifo.m
Matrix size is 9
```

Bode plots are generated in the same way, with one small exception.

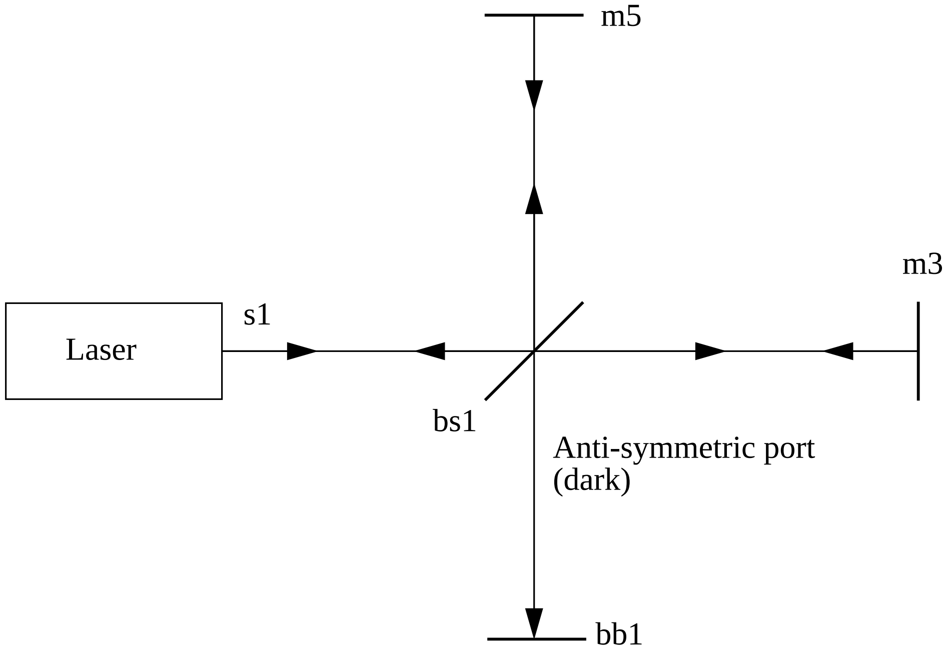
```
ClearBuff;
npoints = 20;
startfreq=10;
stopfreq=10000;
shake[m2,1];
```

```
outindex=index[s1,1,1];
CalcAndPlot[1,outindex,{rf1,3rf1}]
```

The command `CalcAndPlot` takes an optional third argument, which is a list of frequencies to demodulate at. If no argument is given in the third position, *twiddle* assumes all frequencies are equidistant and that the demodulation occurs at the difference frequency. If a demodulation frequency list is given in the third position, then *twiddle* checks that it performs the correct cross products of fields. In this case, the first demodulation at `rf1` will beat the -2 sideband against the -1 sideband, the -1 sideband against the carrier, etc. The `3 rf1` demodulation will beat the -2 sideband against the +1 sideband, and the -1 sideband against the +2 sideband. The output here is two transfer functions, one for each demodulation of the light returning to the source, inphase.

## 2.2 Section 2 Michelson interferometer

In this section, we will go through the steps to generate the *ifo* notebook for a Michelson interferometer. We will asymmetrize the arms to allow the sidebands to interfere constructively at the antisymmetric port, while the carrier is on a dark fringe there. We will also assume the simple case of a single phase modulated sideband. A sketch is helpful:



We've had to add a "beam dump" at the antisymmetric port, which is simply an end mirror with 0 reflectivity, which allows us to make a connection to the beamsplitter from the antisymmetric port. *twiddle* will issue an error message if it encounters an interferometer with un-connected inputs.

The first step is to open a new notebook, and change the input cell to an initialization cell by “Cell->Cell Properties”[1]. When the notebook is saved, this will automatically generate the .m ini-

1. This may be a platform-dependent procedure. Another setting that seems to be required is under “Edit->Preferences->Notebook Options->File Options->AutoGeneratedPackage->Automatic”. The default is Manual, which doesn’t seem to work too well (under Xwindows).

tialization file, which is what *twiddle* looks for. Next, we generate the optical parameters for the elements laid out in the diagram above. We can ignore the beam dump, since later we will give it 0 reflectivity and 0 transmissivity. Since we typically know the transmission of the mirrors, and can estimate the losses, we first say

```
loss = 100. 10^-6
cT2 = .49995
cT3 = .03
cT5 = cT3
```

We can then define functions in Mathematica that calculate the relevant quantities. `other[cT_] = 1. - loss - cT` is a function which will calculate the reflectivity of a mirror as a function of it's transmissivity, assuming the loss is the same for all components. In this definition, `cT` is local to the function. Any number can be put in this slot. `loss` is a global parameter for all optical elements. If we wanted to specify individual losses for each component, we could do this by assigning loss numbers to mirrors as we did transmissivity, then modify `other` to recognize `loss` as a local variable, `other[cT_, loss_] = 1. - loss - cT`. The next step is to use `other` to assign values of reflectivity to each mirror:

```
cR2 = other[cT2]
cR3 = other[cT3]
cR5 = other[cT5]
```

The necessary amplitudes are then calculated by

```
t2 = Sqrt[cT2]
t3 = Sqrt[cT3]
t5 = Sqrt[cT5]
r2 = Sqrt[cR2]
r3 = Sqrt[cR3]
r5 = Sqrt[cR5]
```

It should be noted that all *twiddle* needs are the amplitude reflectivities and transmissivities. How you calculate or assign them is entirely up to you.

We assign the modulation depth and modulation frequency:

```
gamma = 0.1

mfreq = 12.4914 10^6 (* Wavelength of 24 meters *)
```

We now need to build the `props` matrix. We will need to appeal to the sign conventions defined for the optical components. Since we aren't concerned with what happens between the source and the beam splitter, we can fill our first column with 1's. We want to keep the carrier dark at the antisymmetric port while allowing the sidebands to propagate to the antisymmetric port. If the two carriers propagating down the two arms both accumulate the same amount of phase, modulo $\pi$ for a one-way trip, they will interfere destructively at the antisymmetric port. This is due to the sign convention of the beamsplitter, which adds an additional $\pi$ phase on the reflection of the bs1 to m3 carrier to the beam dump, The bs1 to m5 carrier experiences no such phase shift, so the two beams cancel, The phase of a sideband is simply the phase of the carrier plus the phase of the modulation frequency $2\pi\frac{l\nu_0}{c} + 2\pi\frac{l\nu_{mod}}{c} = \phi_{carrier} + \phi_{modulation}$, where $l$ is the length of the arm and the $\nu$'s are the carrier and modulation frequencies. This shows that the sideband phases will depend on our choice for carrier phase. We want the difference of the phases from the two arms at the antisymmetric port to be some integral multiple of $2\pi$. A little algebra shows

$\frac{(l_1 - l_2)\nu_{mod}}{c} = \frac{2n-1}{4}$, $n$ integer, $l_1$ and $l_2$ are the lengths of the two arms. For the specified modulation frequency of 12.4914 MHz, the smallest asymmetry (n = 0) is 6 meters, or π/2 phase. So we know that the phase of the lower and upper sidebands in the shorter arm will be π/2 less than the corresponding phase in the longer arm. We also know that the sidebands are symmetric about the carrier. Specify, for example, that the carrier accumulates π phase propagating down both arms. Then the phase of the sidebands for the long arm will be $\pi \pm \pi/4 \pm \phi_{const}$, where $\phi_{const.}$ is a phase associated with the average arm length, which we can arbitrarily set to 0 here, as long as we're consistent when we define the lengths of the cavities (that is, the average length of the cavities should be a multiple of the modulation wavelength, so $\phi_{const.}$ will be a multiple of 2π). We can now fill out the first and third rows of the second column with Exp[-I 3π/4] and Exp[-I 5π/4]. A little algebra shows that the sideband phases in the second, shorter arm will then have propagators of Exp[-I 5π/4] and Exp[-I 3π/4], respectively. Since we're also not interested in the phase accumulated to the bb1 mirror, those propagation phases can also be set to 1. We then have:

```
props = Transpose[{{       1.      ,      1.     ,       1.      },
                   { Exp[-3 pi/4 I],    -1.     , Exp[-5 Pi/4 I]},
                   { Exp[-5 Pi/4 I],    -1.     , Exp[-3 Pi/4 I]},
                   {       1.      ,      1.     ,       1.      }}]
```

where we have used Mathematica's `Transpose` command to aid in clarity. The first row shows the connection from source to bs1, the second row is bs1 to m3, third row is bs1 to m5, and the last row is bs1 to bb1. The columns correspond to lower sideband, carrier, and upper sideband. A last note concerns the option of specifying the length of the optical paths with the `connect` command. This will check the consistency of the propagation phases of the sidebands with respect to the carrier, given the modulation frequency. The utility of this method of specifying propagation phases is that it is sufficient to define the length of the arms with respect to the modulation wavelength, which is typically a nice large number, 24 meters in our case. Our example has the upper sideband modulation phase at +π/4 in the bs1 to m3 arm, and -π/4 in the bs1 to m5 arm. This determines the length of the bs1 to m3 arm as an integer number of modulation wavelengths plus 3 meters. Obviously the bs1 to m5 arm length is the modulation wavelength minus 3 meters. One final cautionary note: the calculation of the phase using the length of the cavity uses the value of c as $2.99792458 \times 10^8$ m/s. You'll find that the calculation of lengths and/or frequencies needs fairly high accuracy to satisfy the $1°$ criteria.

The last part of the *ifo* notebook labels and connects up all the elements.

```
s1 = source[gamma]
bs1 = beamspl[r2,t2]
m3 = endmirror[r3,t3]
m5 = endmirror[r5,t5]
bb1 = endmirror[0.,0.]
```

gives labels to each of the optical components, and specifies the necessary parameters. Note that we've given the bb1 (beamblock) mirror 0 reflectivity and 0 transmissivity, to ensure that no light enters port 4 of the beamsplitter. Connecting the elements must be done in the order of the columns of props:

```
connect[s1,1,bs1,1,0.]
connect[bs1,3,m3,1,3987,mfreq]
```

```
connect[bs1,2,m5,1,3981,mfreq]
connect[bs1,4,bb1,1,0.]
```

Note the asymmetry of 6 meters, which corresponds to the π/2 phase difference of the side-bands in the two arms at 12.4914 MHz. If this number was not correct, *twiddle* would issue a warning. The entire *ifo* notebook is:

```
other[cT_] = 1. - loss - cT
loss = 100. 10^-6
cT2 = .49995
cT3 = .03
cT5 = cT3
cR2 = other[cT2]
cR3 = other[cT3]
cR5 = other[cT5]
r2 = Sqrt[cR2]
r3 = Sqrt[cR3]
r5 = Sqrt[cR5]
t2 = Sqrt[cT2]
t3 = Sqrt[cT3]
t5 = Sqrt[cT5]

gamma = .1
mfreq = 12.4914 10^6

props = Transpose[{{       1.      ,      1.     ,       1.      },
                   { Exp[-3 pi/4 I],    -1.     , Exp[-5 Pi/4 I]},
                   { Exp[-5 Pi/4 I],    -1.     , Exp[-3 Pi/4 I]},
                   {       1.      ,      1.     ,       1.      }}]

s1 = source[gamma]
bs1 = beamspl[r2,t2]
m3 = endmirror[r3,t3]
m5 = endmirror[r5,t5]
bb1 = endmirror[0.,0.]

connect[s1,1,bs1,1,0.]
connect[bs1,3,m3,1,3987,mfreq]
connect[bs1,2,m5,1,3981,mfreq]
connect[bs1,4,bb1,1,0.]
```

Save this file as something like *michelson_ifo.nb* and make sure that the .m file is also generated, in the same directory. This may require a little futzing with Mathematica to get it to behave properly (see footnote, page 10).

We're ready to build the static interferometer in Mathematica now. Open a working notebook, and run

```
SetDirectory["/home/jim/twiddle/twiddle3.0"];
IFOName = "michelson";
sborder = 1;
<<"twiddle.m"
```

This should generate all the same information as in the Fabry-Perot example, except that the matrix size is now 19, indicating that there are 19 fields that Mathematica must solve for.

Generating the Bode plots is just as simple:

```
ClearBuff;
npoints = 20;
startfreq = 10;
stopfreq = 10000;
shake[m3];
shake[m5,-1];
pdout = {index[bb1,1,1],index[s1,1,1]};
CalcAndPlot[2,pdout];
```

Here we're looking at the transfer functions for the two end mirrors moving out of phase with each other at the anti-symmetric port and at the fields returning to the source, in quadrature.

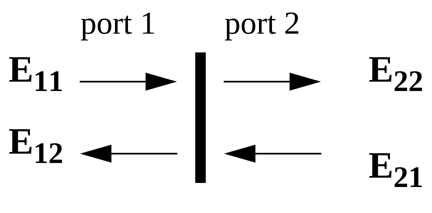
Change the demodulation to inphase (`CalcAndPlot[1,pdout]`). Notice that the first transfer function has a magnitude of -400 dB. This is the default when the signal is 0+0 *i*, since Mathematica can't calculate the argument of 0+0 *i*, and the magnitude $-\infty$ isn't plottable. Does this make sense? Notice the second plot is almost at -400 dB, but obviously this wasn't exactly 0. Does it make sense that there should be a signal here? As you might guess, at this level we're just seeing numerical roundoff errors.

Always remember that it's **important** that, as much as possible, the answers that Twiddle gives you should match your sense for what the answers should be. Examine limit cases of complicated optical configurations to make sure they behave the way you expect them to. Look at the field amplitudes at various points in the interferometer to make sure the carrier and sideband fields are what you expect them to be (using the `seefields` command, see Chapter 4). Never simply trust what a computer tells you.

# 3 INSIDE TWIDDLE

In an analysis where *twiddle* is considering only the first RF sideband on either side of the carrier, it is dealing with light at nine different frequencies: the carrier and its two RF sidebands, and two low-frequency ("audio") sidebands imposed on each of these by the shaking of one or more optical components.

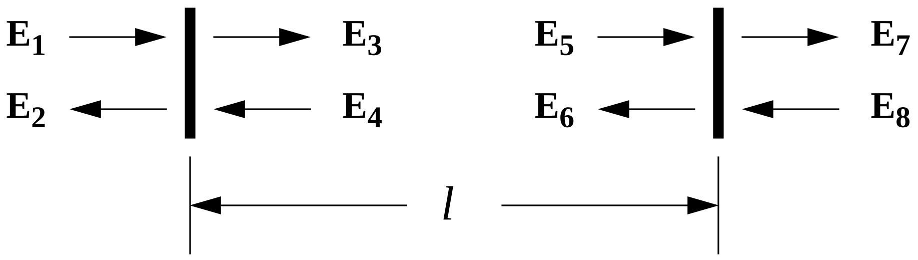
Consider for now a single one of these frequencies. The electric field of this frequency component of the light at some point in the interferometer is related by a set of linear equations to the electric field at other points in the interferometer. For computational convenience we define at each port (a "port" is a place at which light can be incident on the component or reflected from the component) of an optical component an incident field (the electric field due to light incident on that port) and a departing field (the electric field due to light leaving that port). Twiddle defines four fields at each mirror, three fields at an endmirror, eight fields at a beam splitter, and two fields at a source. When the fields are defined this way, the equations relating them are particularly simple. For example, for a mirror,



the electric fields are related by

$$E_{12} = rE_{11} + tE_{21}$$
$$E_{22} = -rE_{21} + tE_{11}$$

where *r* and *t* are the (amplitude) reflectivity and transmissivity of the mirror. The minus sign in the second equation is due to the convention (discussed below) that the reflectivity is negative at port 2 of a mirror. As a second example, suppose that two parallel mirrors are separated by a distance *l* as shown below.

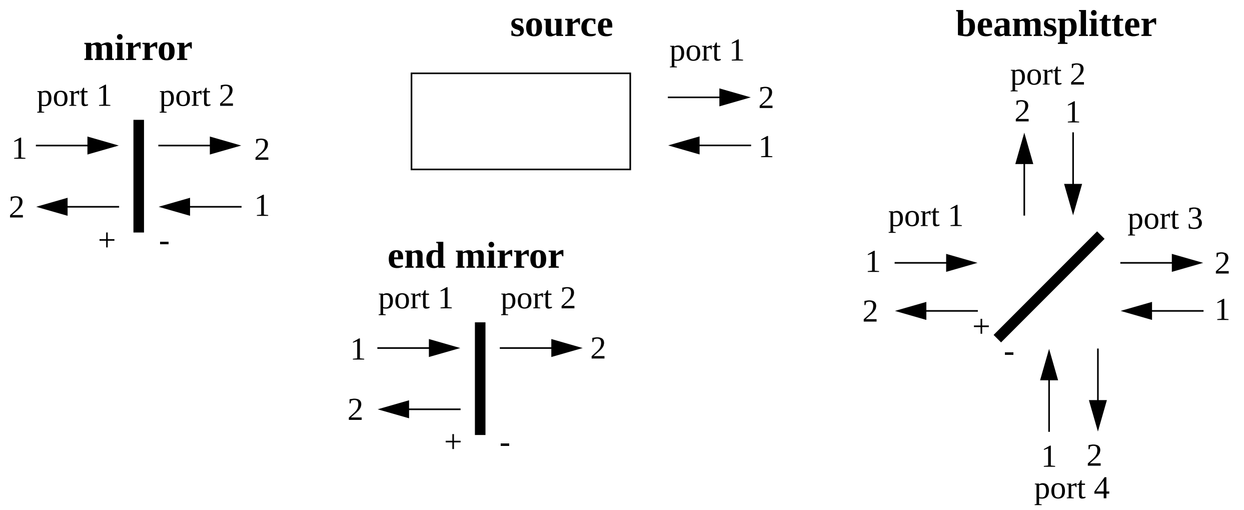


Then the equations relating the incident field at one mirror to the departing field at the other are

$$E_5 = E_3 e^{-i(kl)}$$
$$E_4 = E_6 e^{-i(kl)}$$

where $i = \sqrt{-1}$ and *k* is the wave number corresponding to the frequency being considered. For the RF frequencies in the problem, the factor $e^{-i(kl)}$ must be supplied by the user as an element of *props*, since otherwise the user would have to specify the length to very high precision. For the audio sidebands, the corresponding factor is calculated by *twiddle* using the length of the connection.

For a mirror or beam splitter, it is assumed that the beam reflected from one side of the component experiences a phase reversal upon reflection, whereas the beam reflected from the other side does not. The diagrams below indicate (with a "-") at which port the phase reversal occurs.



From the commands in the *ifo* file specifying the interferometer construction, *twiddle* constructs a matrix which, when inverted and multiplied by the appropriate source vector, gives the electric fields at each component in the interferometer,

A similar matrix is set up for each of the frequencies of interest, To find the transfer function to demodulator output voltage, *twiddle* then simply takes the appropriate sum of products of the field components at each frequency.

The transfer function generated by Twiddle has dimensions of $V(2\pi/\lambda)$, assuming 1W of incident power, a photodetector efficiency of 1V/W and a demodulator which produces an output from its input according to $V_{out}(t) = \frac{\omega_{mod}}{2\pi} \int_{t}^{t+\frac{2\pi}{\omega_{mod}}} V_{in}(t')\cos(\omega_{mod}t')dt'$ (for inphase demodulation; the integral contains $\sin(\omega_{mod}t')$ for quadrature phase demodulation).

# 4 COMMANDS AND FUNCTIONS

This chapter contains a list of commands and functions available in *twiddle* for the construction and analysis of interferometer models.

## 4.1 Model Construction

The following commands are available. The commands `mirror`, `endmirror`, `source` and `beamspl` must be preceded by a statement assigning a matrix to the variable `props`

### props

Syntax:

```
props = {{p-n,1  ,p-n,2  , ... ,p-n,k  },
          {p-n+1,1,p-n+1,2, ... ,p-n_1,k},
           ...
          {pn,1  ,pn,2   , ... ,pn,k   }}
```

(where $p_{i,j}$ is the i$^{th}$ sideband propagator for the j$^{th}$ connection).

`props` contains information about how each RF frequency propagates within each connection. $p_{i,j}$ is the complex constant by which the field of the light entering the connection must be multiplied to find the field of the light leaving the connection at its other end. A sideband propagator may be set to 0 for a particular connection only if the associated `connect` command is given the list of frequencies so that it can calculate the sideband propagator and insert it in place of the 0. The carrier propagators may never be set to 0. See `connect`.

### beamspl

Syntax:

```
name = beamspl[r,t]
```

assigns to *`name`* a component number, and generates a corresponding component: a beamsplitter with (amplitude) transmission *`t`* and reflectivity *`r`*.

## mirror

Syntax:

`name = mirror[r,t]`

assigns to *`name`* a component number, and generates a corresponding component: a mirror with (amplitude) transmission *`t`* and reflectivity *`r`*. In *twiddle*, a mirror may be considered a special case of a beam splitter, where the reflected beams fall back onto the incident beams.

## endmirror

Syntax:

`name = endmirror[r,t]`

assigns to *`name`* a component number, and generates a corresponding component: a mirror with (amplitude) transmission *`t`* and reflectivity *`r`*. The only difference between a mirror and an endmirror is that *twiddle* assumes that there is no light incident on the back of an endmirror.

## source

Syntax:

`name = source[gamma]`

assigns to *`name`* a component number, and generates a corresponding component: a source of light as an input field to the interferometer. The argument *`gamma`* is either

- a scalar variable, in which case it's assumed to be the modulation index governing the phase modulation of the input light. The order to which the phase modulation expansion (in other words, the number of sidebands included in the model) is carried out is specified by defining `sborder` (the default value for `sborder` is 1) before *twiddle* is executed.

or it's

- a vector, in which case it's assumed to be a list of complex amplitudes for the arbitrary sideband configuration on the input light. The ordering of amplitudes should correspond from the lowest frequency sideband to the highest frequency sideband. The carrier, then, should be the middle amplitude (as many sidebands below as above). A list of frequencies corresponding to these amplitudes is given elsewhere.

Source absorbs any light returning to it.

## connect

Syntax:

`connect[nameA, portA, nameB, portB, length(, modfreq)]`

generates a connection from *`portA`* of component number *`nameA`* to *`portB`* of component number *`nameB`*, assuming a distance of *`length`* between them. *`modfreq`* is an optional argument which if specified (equal to the modulation frequency in Hz) will check the sideband phases indicated in `props` with the phase calculated using *`length`* the frequency (or frequencies) indicated in *`modfreq`*. Also, if the corresponding `props` element is zero, it will insert the phase calculated using *`length`* the frequency (or frequencies) indicated in *`modfreq`*. See also `props`.

## 4.2 Analysis

The following commands are used for analysis.

### shake

Syntax:

```
shake[name(, phasor)]
```

tells *twiddle* which component(s) should supply the input motion, from which the transfer function is calculated. *twiddle* then considers this component as a source of audio sidebands on either side of each RF sideband. The optional second argument specifies the amplitude and phase at which to shake the component. A real number should always be used here, otherwise a transfer function with a phase which at DC is neither $0°$ nor $180°$ will result. This second argument may be used:

- to shake two components simultaneously in opposite phase, as in the Michelson example above;
- to correct for the angle of a beam splitter. Without a second argument, `shake` will shake a beam splitter by an amount such that the optical path length change for the reflected beams is the same as it would be for a mirror. This is nearly correct for a beamsplitter operating at nearly normal incidence, and is too large a displacement by a factor of $\sqrt{2}$ for a beam splitter operating at $45°$, since the motion of the beamsplitter affects the optical path length on incidence, and not on reflection, unlike normal incidence mirrors.

### index

Syntax:

```
index[name, port, direction]
```

returns the index of the elements within the vectors of fields (where one vector corresponds to each frequency and one element within each vector to each location in the interferometer) corresponding to the location `name, port,`and `direction`. This command is mainly used by the user inside the commands `CalcAndPlot,CalcAndWrite`, and `seefields`.

### demodin, demodqu

Syntax:

```
demodin[index]
demodqu[index]
```

returns the (complex) value of the transfer function from the displacement of the optical component(s) specified in `shake`, to the output of the inphase or quadrature phase demodulator at the location specified by `index`. It is important to note that in *twiddle* there is no implicit interaction between the photodiodes and the light in the interferometer. A photodiode is modelled as an element which measures the total intensity in the light travelling in a given direction at a given point, without attenuating this light. The user may want to include pickoffs and photodiodes explicitly (in the form of low-reflectivity beamsplitters and zero-reflectivity end mirrors), but this will slow execution. Losses due to pickoffs can be included more efficiently in `props`.

## seefields

Syntax:

`seefields[`*`index`*` (, `*`precision`*`)]`

returns a table of numbers three columns wide, corresponding to the fields at the location specified by *`index`*. Each entry in the table contains a pair of numbers: the real and imaginary parts of one of the fields. The three columns correspond to the lower audio, RF, and upper audio sidebands respectively; the rows correspond to RF frequencies, from lowest to highest. In actuality, the audio fields in the first and third columns represent gain terms, rather than actual light powers, and only the second column represents real powers. This is due to the fact that *twiddle* is calculating the amount of light generated by motion corresponding to 1/*k*, where *k* is the wave vector, in order to scale the transfer function appropriately.

## CalcAndPlot

Syntax:

`CalcAndPlot[`*`demod`*`,`*`outindexlist`*`(,`*`demodulation list`*`)]`

calculates the fields at all points in the interferometer, and demodulates at the indicated outputs specified by *`outindexlist`*, governed by the *`demod`* argument. *`demod`* = `1` does an inphase demodulation, while *`demod`* = `2` does the quadrature demodulation. If *`demod`* does not equal either of these numbers, the program is stopped. `CalcAndPlot` generates a list of the number of frequency points solved for, modulo (npoints/5), while the program is running, which gives the user an idea of the time required to complete the computation. `CalcAndPlot` then outputs Bode plots of the magnitude and phase transfer functions. The optional argument *`demodulation list`* is used to specify frequencies at which to demodulate, when multiple input frequencies are used. If no argument is passed to *`demodulation list`*, then all frequencies are assumed equidistant, and the cross product is made of all fields.

## CalcAndWrite

Syntax:

`CalcAndWrite[`*`outindexlist`*`(,`*`demodulation list`*`)]`

serves the same function as does `CalcAndPlot`, but the output is written to an ASCII text file. Both demodulations are calculated. Data is written in column form. The first column corresponds to the frequency list. The next four columns correspond to inphase magnitude and phase, and quadrature magnitude and phase (magnitude in dB, phase in degrees). This is repeated, looping over *`outindexlist`*, then repeated again for the second frequency in *`demodulation list`*. The name of the file is given by the suffix to the *ifo.m* file, with *.dat* as the extension.