Twiddle

(Formerly *ifresp*)

A Program for Analyzing Interferometer Frequency Response

Version 2.2

Martin W. Regehr

James E. Mason

California Institute of Technology

4/22/96

LIGO-T960079–00–R

Chapter 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 phase modulated at a single frequency. The output voltages are assumed to be derived by sinusoidal demodulation 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.

Some other types of interferometer can also be analyzed with this program. For example, an interferometer illuminated with two colors of light, each phase modulated at two different frequencies, can be analyzed by running the program several times with different parameters, provided that none the phase modulation frequencies is a multiple of any other (to within the signal bandwidth of interest).

Twiddle consists of five pieces of Mathematica code: *Twiddle* and four subroutines called by Twiddle: *operations, components, ifo* and *plot.* The latter two are intended to be modified 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. *plot* contains commands for plotting the response of the interferometer outputs to motions of its components.

During a normal session with the program, the user will first construct a file containing the specifications of the interferometer. This is the *ifo* file. The specifications consist of information about the reflectivity, transmission, and relative spacing of the optical components. Next the user will modify or generate a *plot* file, which specifies the transfer function to be plotted (i.e. from which input to which output). Finally, when run, the *Twiddle* program will produce Bode plots of the transfer function.

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 files, *ifo* and *plot*, 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 files *ifo* and *plot*. 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.

Chapter 2 Tutorial

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

Section 1 The Fabry-Perot Cavity

We will model a Fabry-Perot cavity on resonance. Assumptions include making the carrier resonant, and the sidebands exactly anti-resonant. Implicit in this assumption is that the second order sidebands are negligible (weak modulation limit). The *ifo* and *plot* files for this interferometer are called *fp_ifo* and *fp_plot*. These files should be copied into files with the names *ifo* and *plot*, respectively. Start up Mathematica. *Twiddle* and all its associated files should exist in the directory in which Mathematica is run, or should be in Mathematica's path. Inside Mathematica, all that is needed to run *Twiddle* is to execute the command `<<twiddle`

If you run *Twiddle* using the supplied files, you should first see an incrementing set of numbers printed to the screen. These indicate the number of frequency points evaluated by the program, and let the user estimate how long it will take to complete a given plot. Then a pair of Bode plots should appear, one for the amplitude response (in dB) and one for the phase response (in degrees), both plotted against the log of the frequency. These plots represent the transfer function from the position of the back mirror to the inphase demodulator voltage.

Let's have a look inside *fp_ifo* and *fp_plot*. Here is *fp_ifo*:

```
(* 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]
amp[P_] = Sqrt[P]

r1 = amp[cR1]
r2 = amp[cR2]

t1 = amp[cT1]
t2 = amp[cT2]

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

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

s1 = source[gamma]
m1 = mirror[r1,t1]
connect[s1,1,m1,1,0.]
m2 = endmirror[r2,t2]

connect[m1,2,m2,1,3990., 12.5 10^6]
```

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 propagation phase, 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. `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 $3\pi/2$ phase (Exp[-I $3\pi/2$] = 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 $\pi/2$, 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 are valid. The default for the number of RF sidebands considered is two (one on each side of the carrier). This can be changed by typing `order = ` *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.

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

defines a mirror of amplitude reflectivity `r1` and transmission and `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 beam splitter (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 14 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[m1,2,m2,1,3990., 12.5 10^6]
```

connects port 2 of mirror 1 to port 1 of mirror 2, with an intervening space of 3990 m. 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 elements of `props` corresponding to the sidebands differ by more than 1% from the values calculated using the carrier element and the length of the connection. This is the consistency check mentioned earlier.

Now let's look at *fp_plot*.

```
shake[m2]

outindex = index[s1,1,1] (* demodulate light returning to the source *)

npoints = 50
startfreq = 1
stopfreq = 10000

(* specify if demodulation is done inphase or quadrature:
   demod = 1 is inphase, demod = 2 is quadrature *)

demod = 1


calcandplot[demod]
```

The first line

```
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. The second line

```
outindex = index[s1,1,1] (* demodulate light returning to the source *)
```

tells *Twiddle* where, in the optical topology, to demodulate the light. This is equivalent to placing a pickoff where we wish to find the transfer function. The arguments of the function `index` refer to the component, 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 14 for the definition of a source). The lines

```
npoints = 50
startfreq = 1
stopfreq = 10000
```

establish the number of points to plot (logarithmically spaced in frequency), and the start and stop frequencies in Hz. The assignment `demod = 1` indicates that we wish to look at the inphase demodulated signal, rather than the quadrature demodulation. The function `calcandplot[demod]` then carries out the necessary calculations, and generates the Bode plots of the transfer function. The command `calcandwrite[demod]` is available to write the data to a file, rather than plot the data on the screen.

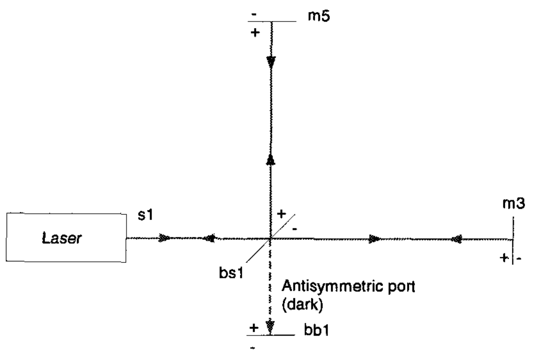
Now let's plot the response of the cavity to high frequency end mirror motions. Modify the third, fourth and fifth lines of *plot* to read

```
npoints = 500
startfreq = 10
stopfreq = 100000
```

and re-run *Twiddle*. 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.

Section 2 Michelson interferometer

In this section, we will go through the steps to generate the *ifo* file 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. 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 steps are to assign the values of the optical parameters. 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]
```

Then we define a function which will calculate the amplitude reflectivities and transmissivities

```
amp[P_] = Sqrt[P].
```

The necessary amplitudes are then calculated by

```
r2 = amp[cR2]
r3 = amp[cR3]
r5 = amp[cR5]

t2 = amp[cT2]
t3 = amp[cT3]

t5 = amp[cT5]
```

We assign the modulation depth and modulation frequency:

```
gamma = 0.1

mfreq = 12.5 10^6
```

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 π 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 π 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 $2\pi\frac{l\nu_0}{c} + 2\pi\frac{l\nu_m}{c} = \phi_{carrier} + \phi_{modulation}$, where l is the length of the arm and the ν'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 2π. A little algebra shows $\frac{(l_1-l_2)\nu_m}{c} = \frac{2n-1}{4}$, n integer, l_1, l_2 are the lengths of the two arms. For the specified modulation frequency of 12.5 MHz, the smallest asymmetry (n = 0) is 6 meters, or $\pi/2$ phase. So we know that the phase of the lower and upper sidebands in the shorter cavity will be $\pi/2$ less than the corresponding phase in the longer cavity. We also know that the sidebands are symmetric about the carrier in the two cavities. 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 $\pi \pm \pi/4 \pm \phi_{\text{const.}}$, where $\phi_{\text{const.}}$ 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 $\phi_{\text{const.}}$ will be a multiple of 2π.). We can now fill out the first and third rows of the second column with Exp[-I $3\pi/4$] and Exp[-I $5\pi/4$]. A little algebra shows that the sideband phases in the second, shorter arm will then have propagators of Exp[-I $5\pi/4$] and Exp[-I $3\pi/4$], 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 arms of the interferometer in 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 wavelenth, which is typically a nice large number, 24 meters in our case. Our example has the upper sideband modulation phase at $+\pi/4$ in the bs1 to m3 arm, and $-\pi/4$ 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 3*10^8 m/s. Using more exact values, such as 2.998*10^8 can generate errors at the 1% level quite easily.

The last part of *ifo* 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 $\pi/2$ phase difference of the sidebands in the two arms at 12.5 MHz. If this number was not correct, *Twiddle* would issue a warning. The entire *ifo* file is:

```
loss = 100. 10^-6

cT2 = .49995
cT3 = .03
cT5 = cT3

other[cT_] = 1. - loss - cT
cR2 = other[cT2]
cR3 = other[cT3]
cR5 = other[cT5]

amp[cT_] = cT^0.5

r2 = amp[cR2]
r3 = amp[cR3]
r5 = amp[cR5]

t2 = amp[cT2]
t3 = amp[cT3]
t5 = amp[cT5]

gamma = 0.1
mfreq = 12.5 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.]
```

The *plot* file is relatively simple. We'll first want to shake the end mirrors m3 and m5 out of phase, then we'll want to look at the signal in port 1 of the beam block from the 1 direction. Furthermore, we'll want this signal in quadrature:

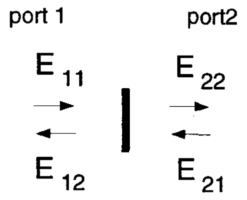
```
shake[m3]

shake[m5,-1]

outindex = index[bb1,1,1]

npoints = 50
startfreq = 1000
stopfreq = 100000

(* specify if demodulation is done inphase or quadrature:
   demod = 1 is inphase, demod = 2 is quadrature *)

demod = 2;

calcandplot[demod];
```

Of particular note here is the second argument of `shake`, which is an optional argument that allows the user to specify relative phases between the motion of components. This argument should be real (see the reference to `shake` in Chapter 4). The user is encouraged to experiment with different parameters and see if the answers agree with intuition. Of special note would be to try an inphase demodulation. Do you expect any signal in this case? Note Mathematica's response.

Chapter 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.

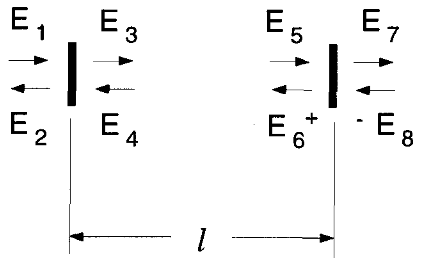
Consider for now a single one of these frequencies. The electric field of this frequency component of the light at some point in the interferomenter is related by a set of linear equations to the electric field at other points in the interferometer. For computational convenience we define at each port (a "port" is a direction in 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 transmission of the mirror. The minus sign in the second equation is due to the convention (dicussed 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.

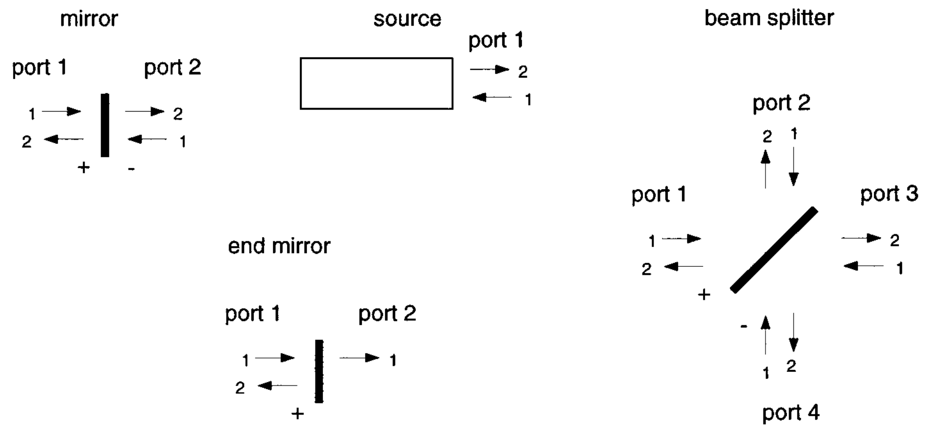


Then the equations relating the incident field at one mirror to the departing field at the other are

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

where $j = \sqrt{-1}$ and k is the wave number corresponding to the frequency being considered. For the RF frequencies in the problem, the factor $e^{-j(kl)}$ 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 efficency of 1 V/W and a demodulator which produces an output from its input according to

$V_{out}(t) = \frac{\omega_m}{2\pi} \int\limits_{t}^{t+\frac{2\pi}{\omega_m}} V_{in}(t') \cos \omega_m t' dt'$ (for inphase demodulation; the integral contains $\sin \omega_m t'$ for quadrature phase demodulation).

Chapter 4 Commands and Functions

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

Section 1 Model Construction

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

propagators

Syntax:

props = {{$p_{-n,1}$, $p_{-n,2}$, $\dots$, $p_{-n,k}$},
{$p_{-n+1,1}$, $p_{-n+1,2}$, $\dots$, $p_{-n+1,k}$},
{$p_{n,1}$, $p_{n,2}$, $\dots$, $p_{n,k}$}}

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

beamspl

Syntax:

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

assigns to *name* a component number, and generates a corresponding component: a beam splitter 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: an end 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, including the carrier and as many RF phase modulation sidebands as specified in the variable `order` (the default value for `order` is 1; it can be changed by assigning `order` a different value before running *Twiddle*). Modulation depth is given by *gamma*. 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 cause the program to generate a warning if sideband propagators have erroneous values.

Section 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*, *direction*. This command is mainly used inside the commands `demodin`, `demodqu`, and `seefields`.

demodin, demodqu

Syntax:

`demodin[`*index*`]`

`demodqu[`*index*`]`

returns the (complex) value of the tranfer 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 beam splitters 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.

calcandplot

Syntax:

`calcandplot[`*demod*`]`

calculates the fields at all points in the interferometer, and demodulates the output 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 10, 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.

calcandwrite

Syntax:

`calcandwrite[`*demod*`]`

serves the same functions as does `calcandplot`, but the output is written to ASCII text files, one each for magnitude and phase. The names of the files are *iamp.dat* or *qamp.dat* and *iphase.dat* or *qphase.dat*, depending on the *demod* argument as in `calcandplot`.

BATCH
START

STAPLE
OR
DIVIDER

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

Twiddle (ver. 2.1)
A Program for Analyzing Interferometer Frequency Response

Martin W. Regehr
James E. Mason

LIGO-T960079-01-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 phase modulated at a single frequency. The output voltages are assumed to be derived by sinusoidal demodulation 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.

Some other types of interferometer can also be analyzed with this program. For example, an interferometer illuminated with two colors of light, each phase modulated at two different frequencies, can be analyzed by running the program several times with different parameters, provided that none the phase modulation frequencies is a multiple of any other (to within the signal bandwidth of interest).

Twiddle consists of five pieces of Mathematica code: *Twiddle* and four subroutines called by Twiddle: *operations*, *components*, *ifo* and *plot*. The latter two are intended to be modified 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. *plot* contains commands for plotting the response of the interferometer outputs to motions of its components.

During a normal session with the program, the user will first construct or modify a file containing the specifications of the interferometer. This is the *ifo* file. The specifications consist of information about the reflectivity, transmission, and relative spacing of the optical components. Next the user will modify or generate a *plot* file, which specifies the transfer function to be plotted (i.e. from which input to which output). Finally, when run, the *Twiddle* program will produce Bode plots of the transfer function.

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 files, *ifo* and *plot*, 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 files *ifo* and *plot*. 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.

2 CHAPTER 2 TUTORIAL

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

2.1 The Fabry-Perot Cavity

We will model a Fabry-Perot cavity on resonance. Assumptions include making the carrier resonant, and the sidebands exactly anti-resonant. Implicit in this assumption is that the second order sidebands are negligible (weak modulation limit). The *ifo* and *plot* files for this interferometer are called *fp_ifo* and *fp_plot*. These files should be copied into files with the names *ifo* and *plot*, respectively. Start up Mathematica. *Twiddle* and all its associated files should exist in the directory in which Mathematica is run, or should be in Mathematica's path. Inside Mathematica, all that is needed to run *Twiddle* is to execute the command `<<twiddle`.

If you run *Twiddle* using the supplied files, you should first see an incrementing set of numbers printed to the screen. These indicate the number of frequency points evaluated by the program, and let the user estimate how long it will take to complete a given plot. Then a pair of Bode plots should appear, one for the amplitude response (in dB) and one for the phase response (in degrees), both plotted against the log of the frequency. These plots represent the transfer function from the position of the back mirror to the inphase demodulator voltage.

Let's have a look inside *fp_ifo* and *fp_plot*. Here is *fp_ifo*:

```
(* 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 (* modulation depth *)

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,m2,1,3990., 12.5 10^6]
```

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 propagation phase, 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. `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 are valid. The default for the number of RF sidebands considered is two (one on each side of the carrier). This can be changed by typing `order = 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`.

```
source[gamma]
```

defines a source of phase modulated light.

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

defines a mirror 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[m1,2,m2,1.3990., 12.5 10^6]
```

connects port 2 of mirror m1 to port 1 of mirror m2, with an intervening space of 3990 m. 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 elements of `props` corresponding to the sidebands differ by more than 1% from the values calculated using the carrier element and the length of the connection. This is the consistency check mentioned earlier.

Now let's look at *fp_plot.*

```
shake[m2]

outindex = index[s1,1,1] (* demodulate light returning to the source *)

npoints = 50
startfreq = 1
stopfreq = 10000

(* specify if demodulation is done inphase or quadrature:
   demod = 1 is inphase, and demod = 2 is quadrature    *)

demod = 1

calcandplot[demod]
```

The first line

```
shake[m1]
```

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. The second line

```
outindex = index[s1,1,1] (* demodulate light returning to the source *)
```

tells Twiddle where, in the optical configuration, to demodulate the light. This is equivalent to placing a pickoff where we wish to find the transfer function. The arguments of the function index refer to the component, 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 13 for the definition of a source). The lines

```
npoints =50
startfreq = 1
stopfreq = 10000
```

establish the number of points to plot (logarithmically spaced in frequency), and the start and stop frequencies in Hz. The assignment `demod = 1` indicates that we wish to look at the inphase demodulated signal. The function `calcandplot[demod]` then carries out the necessary calculations and generates the Bode plots of the transfer function. The command `calcandwrite[demod]` is available to write the data to a file, rather than plot the data on the screen.

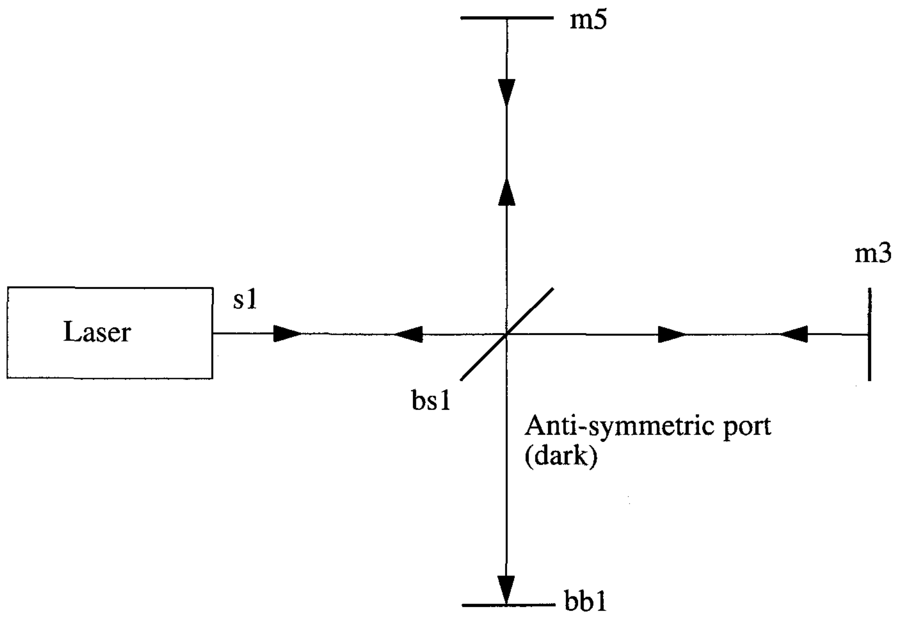
Now let's plot the response of the cavity to high frequency end mirror motions. Modify the third, fourth and fifth lines of plot to read

```
npoints = 500
startfreq = 10
stopfreq = 100000
```

and re-run *Twiddle*. 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.

2.2 Section 2 Michelson interferometer

In this section, we will go through the steps to generate the *ifo* file 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. 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 steps are to assign the values of the optical parameters. 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]
```

We assign the modulation depth and modulation frequency:

```
gamma = 0.1

mfreq = 12.5 10^6
```

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 π 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 π 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 ν'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 2π. 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 12.5 MHz, the smallest asymmetry (n = 0) is 6 meters, or $\pi/2$ phase. So we know that the phase of the lower and upper sidebands in the shorter arm will be $\pi/2$ 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 $\pi \pm \pi/4 \pm \phi_{consti}$, 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 3*10^8 m/s. Using more exact values seem to generate errors at the 1% level quite easily.

The last part of *ifo* 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 $\pi/2$ phase difference of the sidebands in the two arms at 12.5 MHz. If this number was not correct, *Twiddle* would issue a warning. The entire *ifo* file is:

```
(* Twiddle ifo file for a Michelson interferometer     *)
loss = 100. 10^-6

cT2 = .49995
cT3 = .03
cT5 = cT3

other[cT_] = 1. - loss - cT
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 = 12.5 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.]
```

The *plot* file is relatively simple. We'll first want to shake the end mirrors m3 and m5 out of phase, then we'll want to look at the signal in port 1 of the beam block from the 1 direction. Furthermore, we'll want this signal in quadrature:

```
(* Twiddle plot file for a Michelson interferometer      *)
shake[m3]
shake[m5,-1]
```

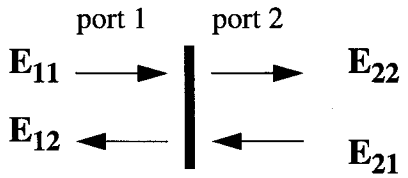
```
outindex = index[bb1,1,1]

npoints = 50
startfreq = 1000
stopfreq = 100000

(* specify if demodulation is done inphase or quadrature:
   demod = 1 is inphase, demod = 2 is quadrature 9

demod= 2
calcandplot[demod]
```

Of particular note here is the second argument of `shake`, which is an optional argument that allows the user to specify relative phases between the motion of components. This argument should be real (see the reference to `shake` in Chapter 4). The user is encouraged to experiment with different parameters and see if the answers agree with intuition. Of special note would be to try an inphase demodulation. Do you expect any signal in this case? Note Mathematica's response.

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.

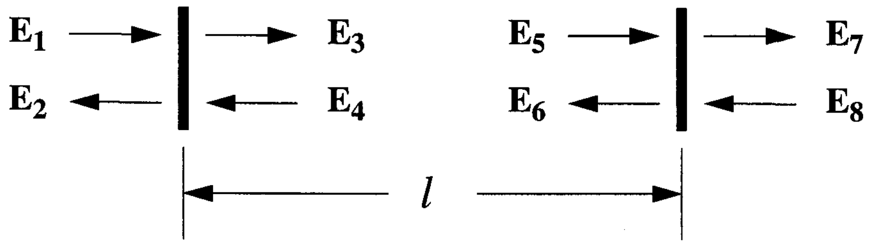
Consider for now a single one of these frequencies. The electric field of this frequency component of the light at some point in the interferometer is related by a set of linear equations to the electric field at other points in the interferometer. For computational convenience we define at each port (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

$$
\begin{aligned}
E_{12} &= rE_{11} + tE_{21} \\
E_{22} &= -rE_{21} + tE_{11}
\end{aligned}
$$

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.

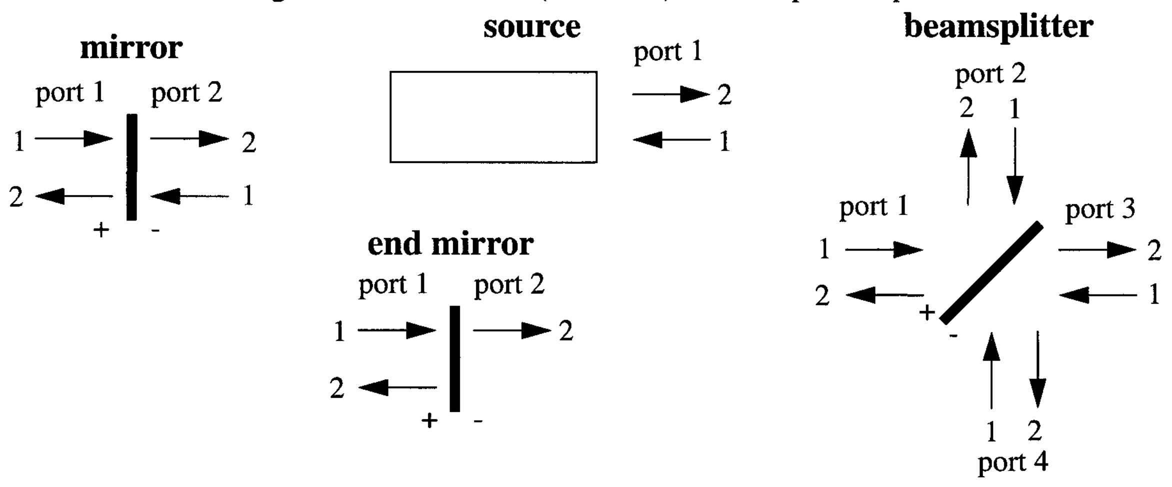


Then the equations relating the incident field at one mirror to the departing field at the other are

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

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`

propagators

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.

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, including the carrier and as many RF phase modulation sidebands as specified in the variable `order` (the default value for `order` is 1; it can be changed by assigning `order` a different value before running *Twiddle*). Modulation depth is given by `gamma`. 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 cause the program to generate a warning if sideband propagators have erroneous values.

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 inside the commands `demodin`, `demodqu`, 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. (Check this).

calcandplot

Syntax:

`calcandplot[demod]`

calculates the fields at all points in the interferometer, and demodulates the output 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 10, 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.

calcandwrite

Syntax:

```
calcandwrite[demod]
```

serves the same function as does `calcandplot`, but the output is written to ASCII text files, one each for magnitude and phase. The names of the files are *iamp.dat* or *qamp.dat* and *iphase.dat* or *qphase.dat*, depending on the *`demod`* argument as in `calcandplot`.