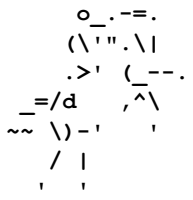


Finesse 2

Frequency domain INterferomEter Simulation SoftwarE

www.gwoptics.org/finesse

19th May, 2014

FINESSE is a fast interferometer simulation program. For a given optical setup, it computes the light field amplitudes at every point in the interferometer assuming a steady state. To do so, the interferometer description is translated into a set of linear equations that are solved numerically. For convenience, a number of standard analyses can be performed automatically by the program, namely computing modulation-demodulation error signals, transfer functions, quantum-noise-limited sensitivities, and beam shapes. FINESSE can perform the analysis using the plane-wave approximation or Hermite-Gauss modes. The latter allows computation of the properties of optical systems like telescopes and the effects of mode matching and mirror angular positions.

FINESSE, the accompanying documentation, and the example files have been written by:

Andreas Freise
School of Physics and Astronomy
The University of Birmingham
Edgbaston, Birmingham, B15 2TT, UK
andreas.freise@googlemail.com

FINESSE has been substantially developed further during the last years with Daniel Brown providing the key contributions, such as the implementation of mirror maps, radiation pressure effects and quantum noise. Daniel's work on the code and the manual was essential for the publication of FINESSE as open source. Charlotte Bond has carefully tested the new code and provided tutorials, examples and documentation.

Parts of the original FINESSE source and 'mkat' have been written by Gerhard Heinzel. Part of the FINESSE source have been written by Paul Cochrane.

This document has been assigned the LIGO DCC number: LIGO-T1300431.

```
------------------------------------------------------------------------
  FINESSE v2.0     - Help Screen -                      18.05.2014
------------------------------------------------------------------------
** Usage (1) kat [options] infile [outfile [gnufile]]
   or    (2) kat [options] basename
   in (2) e.g. basename 'test' means input filename : 'test.kat',
   output filename : 'test.out' and Gnuplot file name : 'test.gnu'.
** Support :
   User support forums:     http://www.gwoptics.org/finesse/forums/
   Online syntax reference: http://www.gwoptics.org/finesse/reference/
** Available options:
 -v : prints version number and build date
 -h : prints this help (-hh prints second help screen)
 -c : check consistency of interferometer matrix
 -max : prints max/min
 -klu-full : switch to KLU solver for parallel frequencies (default)
 -klu      : switch to KLU (Legacy solver)
 --server : starts Finesse in server mode
 --noheader : suppresses header information in output data files
 --perl1 : suppresses printing of banner
 --quiet : suppresses almost all screen outputs
 --convert : convert knm files between text and binary formats
** Available interferometer components:
 l name P f [phase] node                          - laser
 m name R T phi node1 node2                       - mirror
 (or: m1 name T Loss phi ...
      m2 name R Loss phi ... )
 s name L [n] node1 node2                         - space
 bs name R T phi alpha node1 node2 node3 node4    - beamsplitter
 (or: bs1 name T Loss phi ...
      bs2 name R Loss phi ... )
 gr[n] name d node1 node2 [node3 [node4]]         - grating
 isol name S node1 node2 [node3]                  - isolator
 mod name f midx order am/pm [phase] node1 node2  - modulator
 lens f node1 node2                               - thin lens
 sq name f r angle node                           - squeezed input
** Detectors:
 pd[n] name [f1 [phase1 [f2... ]]] node[*]        - photodetector [mixer]
 pdS[n] name [f1 phase1 [f2... ]] node[*]         - sensitivity
 pdN[n] name [f1 phase1 [f2... ]] node[*]         - norm. photodetector
 ad name [n m] f node[*]                          - amplitude detector
 bp name x/y parameter node[*]                    - plots beam parameters
 cp cavity_name x/y parameter                     - plots cavity parameters
 gouy name x/y space-list                         - plots gouy phase
 beam name [f] node[*]                            - plots beam shape
 qd name f phase node[*]                          - quantum quadrature detector
 sd name f [n m] node[*]                          - squeezing detector
 shot name node[*]                                - shot noise
 qshot[S/N] name n f1 [phase1 [f2...]] node[*]    - quantum shotnoise detector
 qnoised[S/N] name n f1 [phase1 [f2...]] node[*]  - quantum noise detector
 pgaind name component motion                     - open loop param. gain det.
** Available commands:
```

```
 fsig name component [type] f phase [amp]         - apply signal
 fsig name component [type] f transfer_func       - signal wth transfer function
 fsig name f                                      - set signal/noise frequency
 fadd f1 f2 f3 ... fN                             - add frequencies to list
 tem[*] input n m factor phase                    - input power in HG/LG modes
 mask detector n m factor                         - mode mask for outputs
 pdtype detector type-name                        - set detector type
 attr component M value Rcx/y value x/ybeta value - attributes of m/bs
 (alignment angles beta in [rad])
 map component filename                           - read mirror map file
 knm component_name filename_prefix [flag]        - save coefficients to file
 smotion component map_file transfer_function     - set surface motion
 maxtem order                                     - TEM order: n+m<=order
 gauss name component node w0 z [wy0 zy]          - set q parameter
 gauss* name component node q [qy] (q as 'z z_R') - set q parameter
 gauss** name component node w(z) Rc [wy(z) Rcy]  - set q parameter
 cav name component1 node component2 node         - trace beam in cavity
 startnode node                                   - startnode of trace
 lambda wavelength                                - overwrite wavelength
 retrace [off|force]                              - re-trace beam on/off
 phase 0-7  (default: 3)                          - change Gouy phases
 (1: phi(00)=0, 2: gouy(00)=0, 4: switch ad phase)
 conf component_name setting value                - configures component
 vacuum components_names                          - specific quantum noise
 tf name factor phase [{p/z f Q [p/z f2 Q2 ...]]  - f,Q transfer function
 tf2 name factor phase [p1,p2,...] [z1,z2,...]    - complex transfer function
** Plot and Output related commands :
 xaxis[*] component param. lin/log min max steps  - parameter to tune
 x2axis[*] component param. lin/log min max steps - second axis for 3D plot
 noxaxis                                          - ignore xaxis commands
 const name value                                 - constant $name
 var name value                                   - tunabel variable $name
 set name component parameter                     - variable $name
 func name = function-string                      - function $name
 lock[*] name $var gain accuracy                  - lock: make $var to 0
 put[*] component parameter $var/$x1/$x2/$fs/$mfs - updates parameter
 noplot output                                    - no plot for 'output'
 trace verbosity                                  - verbose tracing
 yaxis [lin/log] abs:deg/db:deg/re:im/abs/db/deg  - y-axis definition
 scale factor [output]                            - y-axis rescaling
 diff component parameter                         - differentiation
 deriv_h value                                    - step size for diff
** Auxiliary plot commands :
 gnuterm terminal [filename]                      - Gnuplot terminal
 pyterm terminal                                  - Python terminal
 pause                                            - pauses after plotting
 multi                                            - plots all surfaces
                                                    save/load knm file
 GNUPLOT \ ... \ END                              - set of extra commands
                                                    for plotting.
```

```
------------------------------------------------------------------------
 FINESSE v2.0   - Help Screen (2) -                    18.05.2014
------------------------------------------------------------------------
** Alternative calls:
 kat --server <portnumber (11000 to 11010)> [options] inputfile
  starts Finesse in server mode listening on a TCP/IP port
 kat --convert knm_file_prefix 1/2 [new_knm_file_prefix]
  converts binary knm file to ASCII and vice versa
** Some conventions:
 names (for components and nodes) must be less than 15 characters long
 angles of incidence, phases and tunings are given in [deg]
 (a tuning of 360 deg corresponds to a position change of lambda)
 misalignment angles are given in [rad]
** Signal frequency variable
 The variables $fs and $mfs can be used in functions, put commands
 and as the frequency value in any of the detectors. $fs is the positive
 frequency and $mfs is the negative.
** Motion and suspension variables
 Motion names used in xd detectors, slinks and xlinks to select
 particular motions of a suspended optic:
 z = longitudinal motion
 rx, yaw = x-z plane rotation
 ry, pitch = y-z plane rotation
 sN = The Nth surface motion set with the smotion command, from 0 -> N-1

 To set the suspension transfer function with the attr command, use:
 zmech = Longitudinal motion
 rxmech = yaw motion
 rymech = pitch motion

** Geometrical conventions:
 tangential plane: x, z (index n), saggital plane: y, z (index m)
 xbeta refers to a rotation in the x, z plane, i.e. around the y-axis
 R<0 when the center of the respective sphere is down beam
 (the beam direction is defined locally through the node order:
 i.e. mirror: node1 -> node2, beam splitter: node1 -> node3)
 beam parameter z<0 when waist position is down beam
** frequency n: 'n' bit coded word, produces the following output:
    frequency 1: print all frequencies computed
    frequency 2: print frequency coupling matrix for each modulator
** trace n: 'n' bit coded word, produces the following output:
    trace 1:   list of TEM modes used
    trace 2:   cavity eigenvalues and cavity parameters like FWHM,
               FSR optical length and Finesse
    trace 4:   mode mismatch parameters for the initial setup
    trace 8:   beam parameters for every node, nodes are listed in
               the order found by the tracing algorithm
    trace 16:  Gouy phases for all spaces
    trace 32:  coupling coefficients for all components
    trace 64:  mode matching parameters during calculation, if
```

```
               they change due to a parameter change, for example
               by changing a radius of curvature.
    trace 128: nodes found during the cavity tracing
** phase 0-7: also bit coded, i.e. 3 means '1 and 2'
    phase 1:   phase of coupling coefficients k_00 set 0
    phase 2:   Gouy phase of TEM_00 set to 0
    phase 4:   'ad name n m f' yields amplitude without Gouy phase
    (default: phase 3)
** bp, possible parameters for this detector:
    w  : beam radius
    w0 : waist radius
    z  : distance to waist
    zr : Rayleigh range
    g  : Gouy phase
    r  : Radius of curvature (phase front) in meters
    q  : Gaussian beam parameter
    g  : Stability parameter
** isol S, suppression given in dB:
    amplitude coefficient computed as 10^-(S/20)
** maxtem O/off : O=n+m (TEM_nm) the order of TEM modes,
    'off' switches the TEM modus off explicitly
** conf:
   This command is used to configure settings of a component
   that are not physical parameters.
  o Modulators
   - conf component numerical_f_couple n
     n = 1 (on - default) or 0 (off)
     Numerically compares all frequencies and couples them if they
     match the modulation frequency or its harmonics
   - conf component print_f_coupling n
     n = 1 (on) or 0 (off - default)
     print which frequencies couple to which at this modulator
     for the carrier and sideband fields
  o Mirors and Beamsplitters
   - conf component integration_method n
     sets the numerical integration method. Cubature refers to a
     self-adapting routine which is faster but less robust
     1 - Riemann Sum
     2 - Cubature - Serial
     3 - Cubature - Parallel (default)
   - conf component interpolation_method n
     sets the interpolation method for the numerical integration of
     surface maps (use NN for maps with sharp edges):
     1 - Nearest Neighbour
     2 - Linear (default)
     3 - Spline
   - conf component interpolation_size n
     sets the size of the interpolation kernel, must be odd and > 0
     (default=3)
   - conf component knm_flags n
```

```
     sets the knm computation flags which define if coeffs are calculated
     numerically or analytically if possible, see below for values of 'n'
   - conf component show_knm_neval 0/1
     sshows the number of integrand evaluations used for the map integration
   - conf component save_knm_matrices 0/1
     if true the knm matrices are saved to .mat files
     for distortion, surface map and the final result
   - conf component save_knm_binary 0/1
     if true the knm and merged map data is stored in a
     binary format rather than ASCII, see --convert option
     for converting between the 2 formats
   - conf component save_interp_file 0/1
     if true a file is written for each knm
     to output each interpolated point. The file
     has 4 columns: x, y, amplitude, phase.
   - conf component save_integration_points 0/1
     if true all points used for integration are
     saved to a file (use this only with the Riemann
     integrator, Cubature can use millions of points!)
   - conf component knm_order 12/21
     changes the order in which the coupling coefficient matrices
     are computed. 1 = Map, 2 = Bayer-Helms
   - conf component knm_change_q 1/2
     specifies the expansion beam parameter q_L
     if 1 then q_L = q'_1 and if 2 then q_L = q_2
** knm flags: 'n' bit coded word, set computation of coupling coeffs
    use with the 'conf compoent knm_flag' command:
    0  : analytic solution of all effects used
    1  : verbose, i.e. print coupling coefficients
    2  : numerical integration if x and y misalignment is set
    4  : numerical integration if x or y misalignment is set
    8  : calculates aperture knm by integration
    16 : calculates curvature knm by integration
    32 : calculates bayer-helms knm by integration
    (default: knm 8)
```

Contents

Chapter 1

Introduction

FINESSE is a simulation program for interferometers. The user can build any kind of virtual interferometer using the following components:

- lasers, with user-defined power, wavelength and shape of the output beam;
- free spaces with arbitrary index of refraction;
- mirrors and beam splitters, with flat or spherical surfaces;
- modulators to change amplitude and phase of the laser light;
- amplitude or power detectors with the possibility of demodulating the detected signal with one or more given demodulation frequencies;
- lenses and Faraday isolators.

For a given optical setup, the program computes the light field amplitudes at every point in the interferometer assuming a steady state. To do so, the interferometer description is translated into a set of linear equations that are solved numerically. For convenience, a number of standard analyses can be performed automatically by the program, namely computing modulation-demodulation error signals and transfer functions. FINESSE can perform the analysis using plane waves or Hermite-Gauss modes. The latter allows computation of the effects of mode matching and misalignments. In addition, error signals for automatic alignment systems can be simulated.

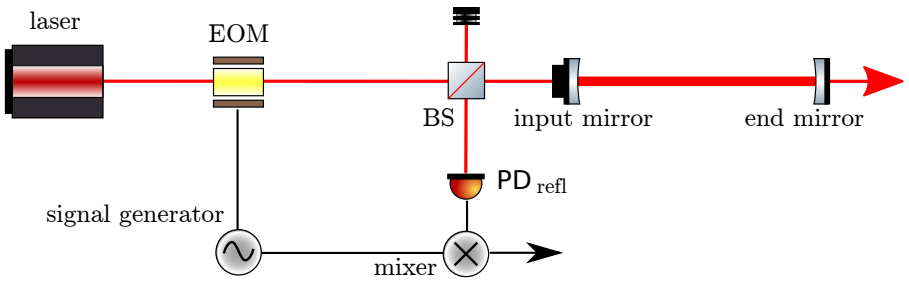


Figure 1.1: *A schematic diagram of a laser interferometer which can be modelled using* FINESSE *(in this case a Fabry-Perot cavity with a Pound-Drever-Hall control scheme).*

Literally every parameter of the interferometer description can be tuned during the simulation. The typical output is a plot of a photodetector signal as a function of one or two parameters of the interferometer (e.g. arm length, mirror reflectivity, modulation

frequency, mirror alignment). FINESSE automatically calls Gnuplot (a free graphics program [Gnuplot]) to create 2D or 3D plots of the output data (alternatively plotting of the output data can be performed with Python or Matlab, see sections 2.6 and 2.7). Optional text output provides information about the optical setup like, for example, mode mismatch coefficients, eigenmodes of cavities and beam sizes.

FINESSE provides a fast and versatile tool that has proven to be very useful during design and commissioning of interferometric gravitational wave detectors. However, the program has been designed to allow the analysis of arbitrary, user-defined optical setups. In addition, it is easy to install and easy to use. Therefore FINESSE is very well suited to study basic optical properties, like, for example, the power enhancement in a resonating cavity or modulation-demodulation methods.

1.1 Motivation

The search for gravitational waves with interferometric detectors has led to a new type of laser interferometer: new topologies are formed combining known interferometer types. In addition, the search for gravitational waves requires optical systems with a very long baseline, large circulating power and an enormous stability. The properties of this new class of laser interferometers have been the subject of extensive research.

Figure 1.2: *Bird's eye view of the GEO 600 gravitational wave detector near Hannover, Germany. Image courtesy of Harald Lück, Albert Einstein Institute Hannover.*

Several prototype interferometers have been built during the last few decades to investigate their performance in detecting gravitational waves. The optical systems, Fabry-Perot cavities, a Michelson interferometer and combinations thereof are in principle simple and have been used in many fields of science for many decades. The sensitivity required for the detection of the expected small signal amplitudes of gravitational waves, however, has put new constraints on the design of laser interferometers. The work of the gravitational wave research groups has led to a new exploration of the theoretical analysis of laser interferometers. Especially, the clever combination of known interferometers has produced new types of interferometric detectors that offer an optimised sensitivity for detecting gravitational waves. Work on prototype interferometers has shown that the models describing the optical system become very complex even though they are based on simple principles. Consequently, computer programs have been developed to automate the computational part of the analysis. To date, several programs for analysing optical systems are available to the gravitational wave community [GWIC].

The idea for Finesse was first raised in 1997, when I was visiting the Max-Planck-Institute for Quantum Optics in Garching, to assist Gerhard Heinzel with his work on Dual Recycling at the 30 m prototype interferometer. We were using optical simulations which were rather slow and not very flexible. At the same time Gerhard Heinzel had developed a linear circuit simulation [Liso] that used a numerical algorithm to solve the set of linear equations representing an electronic circuit. The similarities of the two computational tasks and the outstanding performance of Liso lead to the idea to use the same methods for an optical simulation. Gerhard Heinzel kindly allowed me to copy the Liso source code which saved me much time and trouble in the beginning; and even today many of the Liso routines are still used in their original form inside Finesse.

In the following years Finesse was continually developed during my work at the university in Hannover within the GEO 600 project [Willke01, GEO]. Finesse has been most frequently utilised during the commissioning of GEO 600, some of these simulation results have been published in [Freise03, Lück, Malec, Grote] and in [Freise]. More recently Finesse has been used during the commissioning of Advanced LIGO (see appendix D). Finesse is now actively developed as an open source project and it is now widely used in many other projects; the Finesse home page lists more than 60 documents citing it [gwoptics/impact].

1.2 How does it work?

When the program is run, FINESSE performs the following steps:

Reading a text input file: One has to write an input text file[1] that describes the interferometer in the form of components and connecting nodes (see Section 3.2.1). Several commands specify the computational task and the output format. The command `xaxis`, for example, defines the parameter to be tuned during the simulation.

Generating the set of linear equations: The mutual coupling of all light amplitudes inside the interferometer can be described by linear equations. FINESSE converts the list of components and nodes given in the input file into a matrix and a 'right hand side' vector (see Section 3.1.3) which together represent a set of linear equations. The calculation is initialised by the commands in the input file.

Solving the linear equation system numerically: For each data point, the linear set of equations is updated, the 'right hand side' vector is generated and the system is solved numerically (using a sparse matrix solver such as [Sparse]). This step is repeated for each data point and each possible light frequency (i.e. modulation sidebands).

Writing the data to an output file: After solving the system of equations, all light amplitudes inside the interferometer are known. FINESSE then computes the specified output signals (amplitudes, powers, demodulated signals, etc.) and writes them to a text file (extension '.out'). The screen output is also stored in a file with the extension '.log').

Plotting the data: FINESSE uses external programs to generate and display plots of the output data, it creates a so called batch file with plotting instructions and then calls the external program, by default Gnuplot [Gnuplot]. Gnuplot can display graphs on screen or write the data to a file of a specified graphics format (postscript, gif, etc.) FINESSE creates an additional file (extension '.gnu') that serves as a batch file for Gnuplot and calls Gnuplot to automatically produce the plot. Gnuplot is a free program available for different operating systems. If you do not have Gnuplot installed yet, you should do so (`http://www.gnuplot.info`).

Alternatively you can use Python or Matlab to plot the data. FINESSE saves a Python file (extension '.py') and a Matlab script file (extension '.m'). Python can be called automatically to plot the data similar to the Gnuplot scenario described above. However, plotting the results with Matlab has to be performed by the user: Inside Matlab, change into the working directory containing the 'katfilename.out' and 'katfilename.m' file and call the latter with the command 'katfilename' (replace 'katfilename' by the actual name of the file). By using the command 'gnuterm no' in the input file the automatic call to Gnuplot is suppressed.

[1] There is also a graphical user interface [LUXOR] that can be used to generate the input file.

1.3 Quick start

This section presents some example simulations. FINESSE it is very easy to use, despite this rather voluminous manual. The manual contains much basic information about optical systems and typical tasks in interferometer analysis. Anyone familiar with the analysis of optical systems should be able to install FINESSE and do a first calculation in roughly half an hour. If you do not have much experience with interferometers, you can use FINESSE to learn more about them.

You can find more information online starting at www.gwoptics.org/finesse, such as simple and advanced examples for FINESSE simulations, an online reference page for the FINESSE syntax and advanced installation instructions. Further FINESSE examples in the context of advanced interferometry can also be found in the free online article 'Interferometer Techniques for Gravitational-Wave Detection' [Freise10].

1.3.1 Installation

The installation is simple:

- You only have to copy all files into your working directory. The required files are: **kat** or **kat.exe** (the executable) and **kat.ini**, the initialisation file. In addition, you may want to try the program using my example input files. These have the extension ***.kat**.
- FINESSE is a text-based application that you have **to run from within a terminal or command window**. You can start the program by typing 'kat', which will print a small banner; 'kat -h' will give you a short syntax reference for input files; 'kat -hh' prints further help.
- If you want FINESSE to automatically generate graphical output, you have to have Gnuplot or Python installed and FINESSE must know the correct command to start it. You can add the command for calling Gnuplot and Python on your system to the file 'kat.ini' (see Section 2.2 for more details).
- You should test the program with one of the example files: e.g. 'kat bessel.kat' will cause the program to calculate light field amplitudes behind a phase modulator as a function of the modulation strength (modulation index). The calculated data will be written to 'bessel.out'; also a batch file ('bessel.gnu') for plotting the data with Gnuplot will be created and Gnuplot will be started.
- Please let me know if the above did not work for you!

Jan Harms has created a graphical user interface for FINESSE [LUXOR][2]. This manual does not consider the use of LUXOR but describes the original use of FINESSE, i.e. with ASCII text files and the terminal window. Personally I would only recommend LUXOR for FINESSE beginners. In my opinion the careful use of text files is preferable when the interferometers or the simulation tasks become more complex.

[2] The functionality of LUXOR does not necessarily include the latest features of the current version of FINESSE. Nevertheless, it should be able to handle the majority of simulation tasks.

1.3.2 How to perform a simulation

In order to do a simulation, you have to create a text (ASCII) input file for FINESSE that specifies the interferometer and the simulation task. In the following sections we discuss two example files. The first example is meant for people without much experience in interferometry. It shows some basic syntax without any complicated optics. The second example is aimed at people who know what a 'transfer function' or a 'Pound-Drever-Hall scheme' is and only need to understand the input syntax of FINESSE.

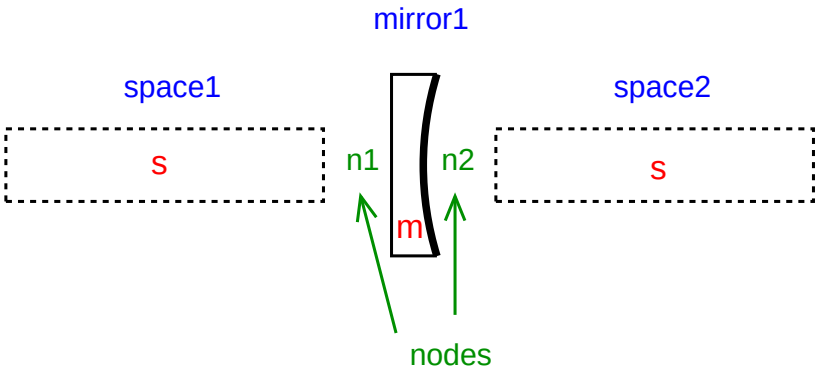


Figure 1.3: *A mirror (m) and two spaces (s) connected via nodes n1 and n2.*

I believe that, before you start writing an input file, it is essential **to first draw the optical setup on a piece of paper**. Next, you have to break down the optical system into its components and nodes (see Section 3.2.1 for details on 'nodes'). The components include mirrors, lasers or 'free spaces', separated by nodes. When, for example, a mirror is next to a 'free space' there is a node between them (see Figure 1.3). All components and nodes have to be given a name. The name will be used to refer to the component, either with a command in the input file, or by FINESSE in warning or error messages.

An input file can consist of a series of component descriptions, commands and some comments (text after '`#`'). Component descriptions are typically entered (one per line) as '`keyword name parameter-list node-list`'. The mirror of Figure 1.3, for example, can be described by

```
m mirror1 0.9 0.1 0 n1 n2
```

The keyword for a mirror is `m`. The name of the component, in this example `mirror1`, is followed by three numerical values. These are the values for the parameters of the component mirror, namely: power reflectivity (R), power transmission (T) and tuning (phi) (see Section 3.3.2 for the definition of 'tuning'). You do not have to memorise all the parameters: calling FINESSE with the command 'kat -h' prints a help screen that includes a short description of the input file syntax with all component parameters. In addition, we provide a handy online reference at `http://www.gwoptics.org/finesse/reference/`.

The last two entries in the component description of the mirror above are the 'node list'. Most components are connected to exactly two nodes. Beam splitters are connected to

four nodes, a laser to one. Every node is connected to at least one component and never to more than two, otherwise the description is inconsistent.

In the component description the keyword specifies the type of component (`m` for mirror, `bs` for beam splitter, etc.), you can choose an arbitrary name but **it must be less than 15 characters long**.

Detectors are special components; they can be located anywhere in the interferometer. Every detector defines one output variable that will be computed by FINESSE. By specifying several detectors, you may compute several signals at the same time.

In addition to component description, commands can be given (one per line). The commands are used to initialise the simulation, they specify what output is to be computed and which parameters are to be changed. For example, the command

```
xaxis space1 L lin 1 10 100
```

defines the x-axis of the output data. In this case it is the length (L) of the component `space1`. The length will be tuned linearly `lin` from 1 to 10 metres in 100 steps. The `xaxis` command has to be given for every simulation. Other commands are optional (like `scale`, which can scale outputs by a user-defined factor). In addition, some commands can be used to customise the graphical output.

The description of the following example input files does not include a detailed explanation of the syntax for commands or component description. Please refer to the syntax reference while studying the following examples. **The help screen (type 'kat -h') gives a short syntax reference. The full syntax reference is given in Appendix G.**

Note: Text from the input files (like commands, keywords, etc.) is printed in a `fixed-width font` throughout this manual.

A simple example: Bessel functions

This example features a laser, a 'phase modulator' (usually an electro-optic device that can modulate the phase of a passing light beam) and 'amplitude detectors'. Amplitude detectors can measure the amplitude and phase of a light field. Such a device does not exist in reality but is a very useful tool in simulations.

The phase modulation of a light field (at one defined frequency, the *modulation frequency*) can be described in the frequency domain as the generation of 'modulation sidebands'. These sidebands are new light fields with a frequency offset to the initial light. In general, a symmetric pair of such sidebands with a frequency offset of plus or minus the modulation frequency is always generated. For stronger modulations, symmetric pairs at multiples of the modulation frequency are also generated.

The amplitude of these sidebands can be mathematically described using Bessel functions (see Section 3.4.3 for details on phase modulation and Bessel functions). In this example, the amplitudes of three modulation sidebands are detected. The result can be used to check whether the implementation of Bessel functions in FINESSE is correct.

The input file 'bessel.kat' for this simulation looks as follows:

```
#-------------------------------------------------------------------
# bessel.kat test file for kat 0.70
#
# freise@rzg.mpg.de  02.03.2002
#
# The "#" is used for comment lines.
#
# Testing the Bessel functions :
#
#               EOM
#             .-----.
#             |     |
#     --> n0  | eo1 |  n1 -->
#             |     |
#             '-----'
#-------------------------------------------------------------------

l i1 1 0 n0                     # laser P=1W f_offset=0Hz
mod eo1 40k .05 5 pm n0 n1      # phase modulator f_mod=40kHz
                                # midx=0.05 order=5
ad bessel1 40k n1               # amplitude detector f=40kHz
ad bessel2 80k n1               # amplitude detector f=80kHz
ad bessel3 120k n1              # amplitude detector f=120kHz
xaxis eo1 midx lin 0 10 1000    # x-axis: midx of eo1
                                # from 0 to 10 (1000 steps)
yaxis abs                       # y-axis: plot absolute
gnuterm x11                     # Gnuplot terminal: X11
```

The only two components of the setup in this example are the laser defined by :

```
l i1 1 0 n0
```

and the modulator:

```
mod eo1 40k .05 5 pm n0 n1
```

These two components are connected via node `n0`, and the exit of the modulator is node `n1`. The laser provides the input field at frequency 0 Hz and a power of 1 W. All frequency values have to be understood as offset frequencies to a default frequency, which can be set by specifying a default wavelength in the init file 'kat.ini'. When the laser beam passes the modulator, sidebands are added at multiples of the modulation frequency. In this case, the modulation frequency is 40 kHz. The strength (or depth) of the modulation can be specified by the modulation index (`midx`); here `midx` =0.5. In general, the sidebands generated by a phase modulation with modulation frequency $\omega_m/2\pi$ and `midx` $= m$ can

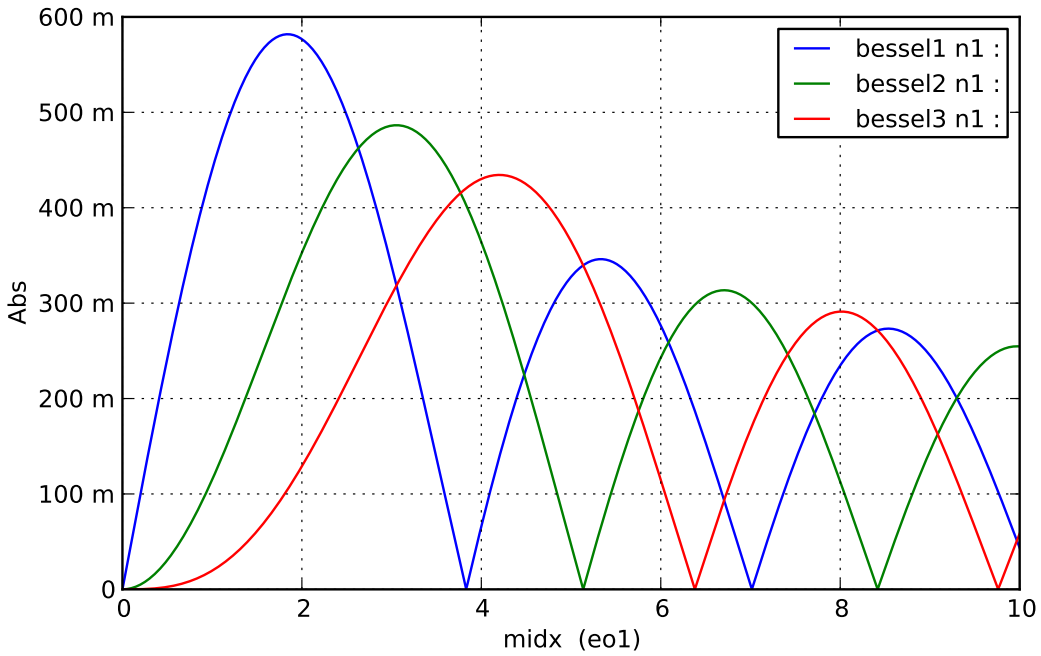


Figure 1.4: *Simple example: testing the Bessel functions. (The plots shown in this manual have been automatically created by* FINESSE *using the Python batch file.)*

be described by the following sum (see Section 3.4.3):

$$\sum_{k=-\infty}^{\infty} i^k \, J_k(m) \, e^{\mathrm{i}\, k\omega_m \, t}, \tag{1.1}$$

with $J_k(x)$ as the Bessel function of order k. For small modulation indices, the Bessel function becomes very small with increasing k. Therefore, usually only a finite part of the above sum has to be taken into account:

$$\sum_{k=-order}^{order} i^k \, J_k(m) \, e^{\mathrm{i}\, k\omega_m \, t}. \tag{1.2}$$

The maximum value for k ('order') is set as a parameter in the component description of the modulator (`order`). In this example, `order` is set to 5 which will result in 11 light fields leaving the modulator: 1 laser field, 5 sidebands with positive frequency offsets (40 kHz, 80 kHz, 120 kHz, 160 kHz, 200 kHz) and 5 sidebands with negative frequency offsets (-40 kHz, -80 kHz, -120 kHz, -160 kHz, -200 kHz). In order to detect some of the light fields after the modulator, we connect 'amplitude detectors' (`ad`) to node `n1`:

```
ad bessel1 40k n1
ad bessel2 80k n1
ad bessel3 120k n1
```

For each detector, a different frequency is specified (40 kHz, 80 kHz and 120 kHz). This means that we will compute field amplitudes for three different sidebands. The setup is now completely described. Next, we have to define the simulation task:

```
xaxis eo1 midx lin 0 10 1000
yaxis abs
```

The compulsory command `xaxis` specifies the parameter we want to tune during the simulation. In this example, the parameter `midx` of the modulator (`eo1`) will be changed linearly (`lin`) from 0 to 10 in 1000 steps. The command `yaxis abs` specifies that the absolute values of the computed complex field amplitudes will be plotted. The command `gnuterm x11` specifies a screen output for Gnuplot in a typical Unix environment. Windows users should use `gnuterm windows`. In addition, there are a number of predefined graphics formats (Gnuplot terminals) for file output (like ps, eps, gif). If no Gnuplot terminal is given FINESSE uses 'x11' on Unix and 'windows' on Windows systems.

In summary, we have set up a very simple optical system which generates phase-modulated sidebands. By running the simulation, we can now compute amplitudes of the sidebands as a function of the modulation index. The resulting plot is shown in Figure 1.4.

A more complex example: A Fabry-Perot cavity in a Pound-Drever-Hall setup

This example consists of two sequential simulations: a) the generation of a Pound-Drever-Hall error signal for a simple Fabry-Perot cavity, and b) the transfer function with respect to this error signal (mirror motion to error signal).

The error signal: pdh-signal.kat The input file is:

```
#----------------------------------------------------------------
# pdh-signal.kat test file for kat 0.70
# (Error signal of the Pound-Drever-Hall signal)
#
# freise@rzg.mpg.de  02.03.2002
#
# The "#" is used for comment lines.
#
#                      m1                              m2
#         .-----.     .-.                             .-.
#         |     |     | |      . . . . . . . . .      | |
# --> n0  | EOM |  n1 | |  n2  .      s1         .  n3  | |
#         |     |     | |      . . . . . . . . .      | |
#         '-----'     | |                             | |
#                     '-'                             '-'
#----------------------------------------------------------------

## reflectivity of first mirror set to 0.9 to get a 'nice' plot
```

```
## the setting is different in 'pdh.kat'!

m m1 0.9 0.0001 0 n1 n2          # mirror R=0.9 T=0.0001, phi=0
s s1 1200 n2 n3                  # space  L=1200
m m2 1 0 0 n3 dump               # mirror R=1 T=0 phi=0

l i1 1 0 n0                      # laser P=1W, f_offset=0Hz

mod eo1 40k 0.3 3 pm n0 n1       # phase modulator f_mod=40kHz
                                 # midx=0.3 order=3
pd1 inphase 40k 0 n1             # photo diode + mixer
                                 # f_demod=40kHz phase=0
pd1 quadrature 40k 90 n1         # photo diode + mixer
                                 # f_demod=40kHz phase=90degrees
xaxis m2 phi lin -90 90 400      # xaxis: tune mirror m2
                                 # from -90 to 90 (400 steps)
yaxis abs                        # plot 'as is'
```

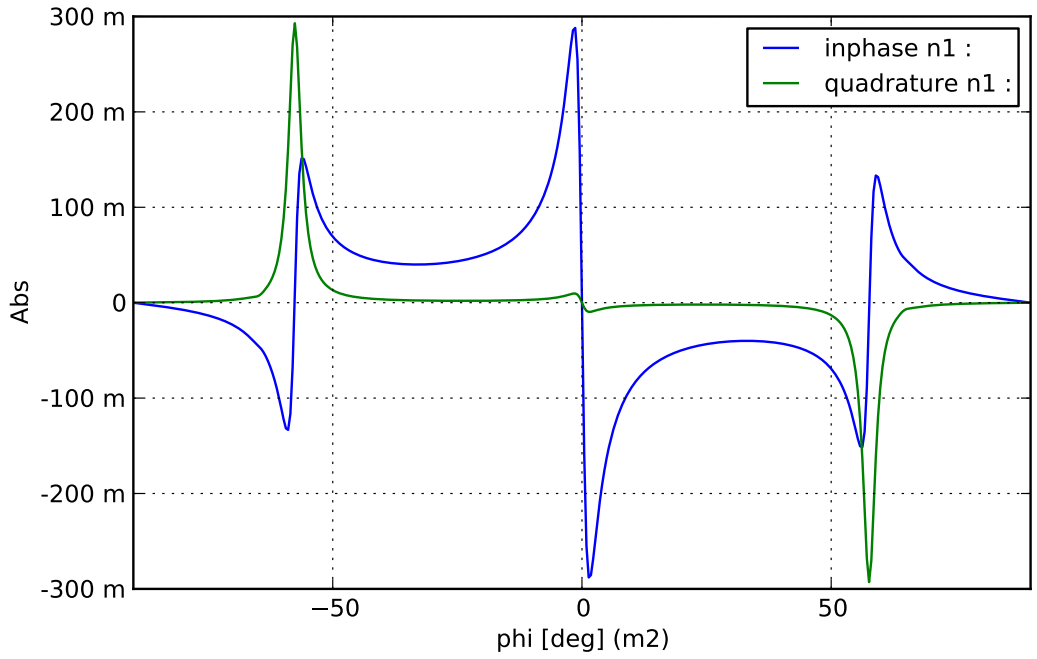


Figure 1.5: *Second example: a Pound-Drever-Hall error signal.*

The interferometer is a simple Fabry-Perot cavity that consists of two mirrors (`m1`) and (`m2`) and a 'free space' (`s1`) in between. The laser (`i1`) provides an input field with a power of 1 Watt. This beam is passed through a modulator which applies a phase modulation. The connecting nodes are `n0, n1, n2, n3`.

The first components define a 1200 m long over-coupled cavity:

```
m m1 0.9 0.0001 0 n1 n2
s s1 1200 n2 n3
m m2 1 0 0 n3 dump
```

The cavity is resonant for the default laser frequency because the two mirror tunings (`phi`) are set to zero (see Section 3.3.2). The default frequency is set by the value 'lambda' in the init file 'kat.ini', the default value for 'lambda' is 1064 nm. The laser:

```
l i1 1 0 n0
```

has an offset frequency of 0 Hz, which means the laser light has the default laser frequency and is thus resonant in the cavity. Next, we need to phase modulate the light (the Pound-Drever-Hall scheme is a modulation-demodulation method):

```
mod eo1 40k 0.3 3 pm n0 n1
```

The Pound-Drever-Hall signal can now be generated with a photodetector and one 'mixer'. A mixer is an electronic device that can perform a demodulation of a signal by multiplying it with a reference signal at the modulation frequency (local oscillator). For demodulation one has to specify a *demodulation phase*. In general, when the optimum phase is not yet known, two components of the signal, 'in-phase' and 'quadrature', are usually computed where the demodulation phase of the 'quadrature' signal has a 90-degree offset to the 'in-phase' demodulation:

```
pd1 inphase 40k 0 n1
pd1 quadrature 40k 90 n1
```

These two detectors detect the light power reflected by the mirror (`m1`) and demodulate the signal at 40 kHz. The typical Pound-Drever-Hall error signal is plotted as a function of the mismatch of laser frequency to cavity resonance. In this example, we choose the `xaxis` to be the microscopic position of the second mirror:

```
xaxis m2 phi lin -90 90 400
```

The command `xaxis` defines the parameter that is varied during the simulation. At the same time, it defines the x-axis of the output (plot). Here, the previously defined mirror (`m2`) is moved by changing the tuning (`phi`) linearly (`lin`) from -90 degrees to 90 degrees in 400 steps. This starts the simulation with the cavity being anti-resonant for the laser light and sweeps the cavity through the resonance until the next anti-resonance is reached. The resulting plot is shown in Figure 1.5.

The transfer function: 'pdh.kat' The second part of this example is very similar to the first part; it also employs a Fabry-Perot cavity with a Pound-Drever-Hall setup. This time, however, we are not interested in the error signal as a function of a mirror position, but in the transfer function of the optical system in a potential feedback loop. We assume that an actuator can move the input mirror (`m1`) and we want to know the transfer function from a displacement of `m1` to the output of the photodetector plus mixer (from the previous example). Please note that the cavity parameters are now different. The input file is:

```
#------------------------------------------------------------------------
# pdh.kat test file for kat 0.99
# (Transfer function of the Pound-Drever-Hall signal)
#
# adf@rzg.mpg.de  07.02.2005
#
#
#                       m1                              m2
#         .-----.     .-.                             .-.
#         |     |     | |      . . . . . . . . . .      | |
# --> n0  | EOM |  n1 | |  n2  .      s1         .  n3  | |
#         |     |     | |      . . . . . . . . . .      | |
#         `-----'     | |                               | |
#                     `-'                               `-'
#------------------------------------------------------------------------

m m1 0.9999 0.0001 0 n1 n2
s s1 1200 n2 n3
m m2 1 0 0 n3 dump

l i1 1 0 n0

mod eo1 40k 0.3 3 pm n0 n1

fsig sig1 m1 10 0

pd2 inphase 40k 0 10 n1

xaxis sig1 f log .01 100 400
put inphase f2 $x1

yaxis db:deg
```

Most of the above is similar to the previous example. Only one new component has been added:

```
fsig sig1 m1 10 0
```

This is the *signal frequency*. It can be understood as connecting the source from a network analyser to an actuator which can move mirror `m1`. This means a periodic signal called `sig1` now 'shakes' the mirror at 10 Hz (phase=0). The periodic movement of the mirror can be described as a phase modulation of the light that is reflected by the mirror, i.e. phase modulation sidebands are generated.

The photodetector used in the previous example to compute the Pound-Drever-Hall error signal has to be extended by another mixer to detect the field amplitude *at the signal frequency*:

```
pd2 inphase 40k 0 10 n1
```

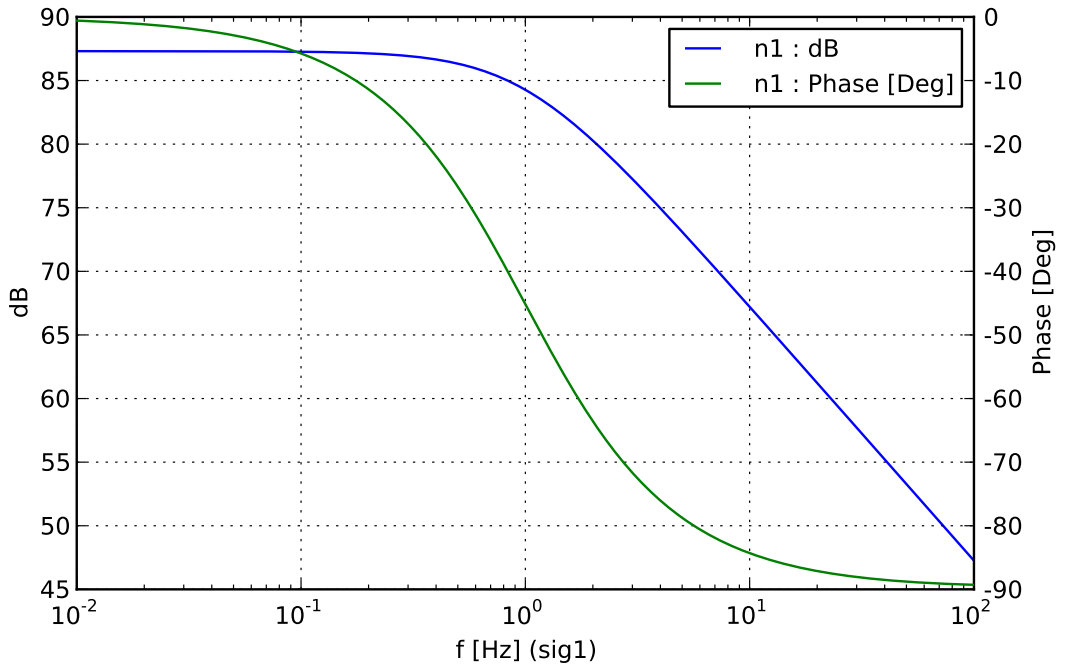


Figure 1.6: *Second example: the transfer function of the a Pound-Drever-Hall error signal (mirror motion to error signal).*

The first demodulation at 40 kHz (demodulation phase 0 degrees) is still the same as before. The second demodulation is at 10 Hz, the signal frequency. You will note that for the second demodulation no demodulation phase is given. If the demodulation phase is not given, the output is (mathematically) simply a complex number representing the amplitude and relative phase of the error signal at the signal frequency[3]. If we now sweep the signal frequency (simultaneously at the source and the second mixer), we will get a transfer function. This can be done by the following commands:

```
xaxis sig1 f log .01 100 400
put inphase f2 $x1
```

The parameter to be swept is `sig1`, the signal frequency. In order to always compute the transfer function, the frequency of the second demodulation at the photodetector must also be changed accordingly. This is assured by the `put` command. `put` sets an interferometer parameter to the value of a variable. In this case it puts the x-axis value $x1$ to the second frequency of the photodetector (`f2`). The resulting plot is shown in Figure 1.6.

[3] In an experiment, this is slightly more complex: a network analyser would perform the demodulation at the signal frequency *twice* with two different demodulation phases and then calculate the amplitude and phase of the signal

Chapter 2

The program files

2.1 kat—the main program

The name of the binary, i.e. the command to start FINESSE is 'kat'. The syntax is:
`kat [options] infile [outfile [gnufile]]`
or
`kat [options] basename`
where e.g. basename 'test' means input filename : 'test.kat', output filename : 'test.out' and Gnuplot batch filename : 'test.gnu' (parameters in square brackets are optional). The input file has to be provided by the user, output and Gnuplot files are created by FINESSE. Available options :

-h : prints first help screen with short syntax reference
-hh : prints second help screen with some conventions
-v : prints version and exits
-c : forces consistency check of interferometer matrix (slow)
-max : prints maximum and minimum of every plot
-klu : forces KLU sparse matrix solver
-sparse : forces SPARSE sparse matrix solver
--noheader : suppresses the printing header information in output data files
--server : starts FINESSE in server mode (see 7.2.2)
--perl1 : suppresses the printing of the banner and Gnuplot command
--convert : converts `knm` files between text and binary formats, see appendix E.
--quiet : suppresses almost all screen outputs

2.2 kat.ini—the init file for kat

The file 'kat.ini' is read when the program is started. It is a text file in which some program parameters can be defined.

For FINESSE to find the 'kat.ini' file you have to either place one copy into the current working directory (i.e. the directory containing the FINESSE input files you are working with), or you specify a global variable 'KATINI' on you computer which contains the full path to a 'kat.ini' file. Please refer to help service of your operating system in order find

out how to set such a variable. On many Unix-like system this can be done by adding some command like:

```
export KATINI=$HOME/work/kat/kat.ini
```

to a configuration file.

The following parameters can be set within the 'kat.ini' file:

clight : speed of light
lambda : main wavelength of the input laser light ('lambda' sets λ_0 and thereby defines ω_0)
deriv_h : step size for numerical differentiation (this parameter can be overwritten in the input file with the same syntax, i.e. `deriv_h value` see also page 220)
qeff : quantum efficiency of photodetectors
epsilon_c : $\epsilon_c = \epsilon_0 \cdot c$ used for relating light power to field amplitudes, $P = \epsilon_c |EE^*|$
n0 : default refractive index for spaces
gnuversion : version number of your Gnuplot binary (a two digit number, for example, 4.2)
PDTYPE : photodetector definition (see also page 206)
GNUCOMMAND : system command to start Gnuplot
GNUTERM : Gnuplot plotting terminal description
PYTHONCOMMAND : system command to start Python
PYTERM : Python plotting terminal description
PLOTTING : selecting default plotting program (Gnuplot/Python)
`quantum_scaling`: Sets how all photodiode quantum noise outputs are scaled: 1 - PSD, 2 - PSD in units of hf, 3 - ASD, 4 - ASD in units of $\sqrt{hf}$. ASD = amplitude spectral density, PSD = power spectral density.

The following parameters can be used for customising the locking algorithm :

locksteps : total number of steps for trying to achieve the locking condition
autostop : if set to 1, stop locking after first failure
sequential : bit coded, 0/1 sequential off/on, 5 first lock sequential. A sequential lock includes a lock hierarchy based on the order of the `lock` commands in the input file. The first lock is kept at zero while the second is changing, the first two are kept locked while the third is changing etc. The sequential lock has proven to be slower but more successful in finding the operating point. Often it is convenient to use sequential locking only for the first data point and then switch to the faster parallel locking in which all loops are iterating together.
autogain : switch for the automatic gain control, 0/1/2 = off/on/verbose
lockthresholdlow : threshold for 'gain too low' check
lockthresholdhigh : threshold for 'gain too high' check
locktest1 : number of steps to wait until loop gain is checked
locktest2 : number of checks to be invalid before the gain is changed
gainfactor : in case of action, change gain by this factor

The '`#`' sign is used for comment lines, and parameters are specified as 'name value',

e.g. 'clight 300000000.0'. If the program cannot read or find the 'kat.ini' file it uses the following default values:

clight : 299792458.0
lambda : 1.064e-6
deriv_h : 1e-3[1]
qeff : 1.0
epsilon_c : 1.0
n0 : 1.0
gnuversion : 4.2
GNUCOMMAND : 'c:\programs\gnuplot\wgnuplot.exe' for Windows systems, 'gnuplot -persist' for Linux systems and '/sw/bin/gnuplot -persist' on OS X.
locksteps : 10000
autostop : 1
sequential : 5
autogain : 2
lockthresholdlow : 0.01
lockthresholdhigh : 1.5
locktest1 : 5
locktest2 : 40
gainfactor : 3

The Gnuplot terminal 'x11' or 'windows' is selected with respect to the operating system. In order to plot the data with Gnuplot you may have to adjust GNUCOMMAND which is the system command used by FINESSE to start Gnuplot. The command must include the full pathname and all options.

Several Gnuplot terminals are predefined in 'kat.ini'. The syntax is as follows:

```
GNUTERM name
(some Gnuplot commands like e.g.:
set term postscript eps
set title
...)
END
```

Please read the Gnuplot manual for information about the Gnuplot commands.

Similarly, if you prefer to create the graphical output with Python you need to adjust the PYTHONCOMMAND and can add details using the PYTERM commands. Finally, the command PLOTTING may be used to define the default plotting, i.e. Gnuplot or Python. Note that in all cases the script files fro plotting with Gnuplot, Python and Matlab will be generated, so that you can conveniently plot the result with any of these program at a later time.

Furthermore, the file 'kat.ini' hosts definitions for photodetector types that have some special features with respect to the detection of Hermite-Gauss modes. Please also see

[1] Note that you have to use a smaller value for 'deriv_h' if you use alignment angles because the angles are typically of the order of 1e-6 and 'deriv_h' must be smaller.

Section 4.8.3 and page 206 for an explanation of these detector definitions. Many different types of real detectors (like split detectors) or (spatially) imperfect detection can be simulated using this feature. The syntax for the type definitions:

```
PDTYPE name
...
END
```

Between `PDTYPE` and `END` several lines of the following format may be given:

1. '`0 1 0 2 1.0`', the beat between TEM_{01} and TEM_{02} is scaled by a factor of 1.0
2. '`0 0 * 0 1.0`', '*' means 'any': the beats of TEM_{00} with TEM_{00}, TEM_{10}, TEM_{20}, TEM_{30}, etc. are scaled by a factor of 1.0
3. '`x y x y 1.0`', '`x`' or '`y`' also means 'any' but here all instances of '`x`' are always the same number (likewise for '`y`'). So, in this example, all beats of a mode with itself are scaled by 1.0

All beat signals not explicitly given are scaled by 0.0. Please take care when entering a definition, because the parser is very simple and cannot handle extra or missing spaces or extra characters. The file 'kat.ini' in the FINESSE package includes the definitions for split photodetectors.

2.3 *.kat—the input files (how to do a calculation)

The program does not work interactively, i.e. all the information about the optical setup and the calculation task has to be stored in one input text file before the program is called. This section describes the syntax of the input files. For a better understanding please also look at the online examples. In addition, Appendix G gives an extensive syntax description. Together with the given examples this should allow one to understand the input file syntax for all possible simulation tasks.

A line of the input file can be empty, specify **one** component, or specify **one** command. Text after a '`#`' sign is treated as a comment. A component entry has the following syntax:

```
component_type name parameter_list node_list
```

Component names and node names must be less than 15 characters long. For example a mirror can be specified by

```
m mirror1 0.9 0.1 0 n1 nout3
```

where '`m`' is the keyword for the component mirror, '`mirror1`' is the name of the component. The parameter list of a mirror is 'power-reflectivity power-transmittance tuning' or in short 'R T phi'. The above example therefore specifies a mirror with R=0.9, T=0.1 and phi=0 connected to nodes '`n1`' and '`nout3`'.

Node names can be chosen by the user and must not be longer than 15 characters. If a special node has only one connection and will not be used for detection either, the special name '`dump`' can be used to indicate a beam dump. This does not

affect the results but reduces the set of linear equations by one and thus speeds up the calculation.

Note that **even if you want to tune (or sweep) a certain parameter you have to enter a fixed value at the proper place first**. Imagine that you have to build the full interferometer before you start moving or shaking things. The commands then follow the interferometer description.

2.4 *.out—the output files

The output files (*.out) are the main output of FINESSE, i.e. they contain the result of the simulation run. These files contain the calculated pure data in text format. The first three lines are a header containing information about the simulation and the output data, a typical header might look like:

```
% Finesse 0.99.8 (3200), 11.06.2008
% 2D plot, y1axis: Abs
% phi [deg] (m1), tr1, tr2
```

The first line contains the version, build number and build date of the FINESSE binary. The second line defines whether the data refers to a 2D or 3D plot and what y-axes have been specified. The third line then gives the labels of the data columns, i.e. the name of the x-axis (or x-axes), in this case 'phi [deg] (m1)', and the names of the detectors, here 'tr1' and 'tr2'.

The '%' sign is used as a comment char in this case because Matlab and Gnuplot can recognise this as a comment char. If your Gnuplot version complains about the header you can try to add the following to your Gnuplot configuration file:

```
set datafile commentschars "#!%"
```

Alternatively you can of course run FINESSE with the `--noheader` option, which suppresses the header in the output files.

The header is followed by the data, stored in rows and columns:

`x y1 [y2 y3 y4 ...]` for 2D plots

`x1 x2 y1 [y2 y3 y4 ...]` for 3D plots

where `x1` is the first x-axis, in 3D plots `x2` is used for the second x-axis. The y values correspond to various graphs, for example:

```
x amplitude1 phase1 amplitude2 phase2
```

2.5 *.gnu—the Gnuplot batch files

These files are batch files for Gnuplot. They are text files with a few simple commands that tell Gnuplot which file it should read and how it should plot it. You can easily change the file yourself to vary the look of the plot or to do some calculations within Gnuplot with the data. Be aware that if you don't rename the file, other runs with the same input file will overwrite the Gnuplot batch file.

2.6 *.m—the Matlab script files

These files are Matlab input files containing the necessary commands to plot the data in the '.out' files with Matlab. To do so, start Matlab, then inside Matlab, change into the working directory containing the 'katfilename.out' and 'katfilename.m' file and call the latter with the command 'katfilename' (replace 'katfilename' by the actual name of the file). Please note that Matlab does not recognise all filenames, for example, you must not use minus or plus signs in Matlab script names. Therefore FINESSE will replace any '-' in the basename by '_' for creating the corresponding name of the Matlab file. Again, please be aware that if you don't rename the file, other runs with the same input file will overwrite the Matlab file.

The Matlab files actually are not scripts but contain function. In order to get more information about using these you can get help by typing `help katfilename` (again replace 'katfilename' with the actual name of the file). This should print something like:

```
------------------------------------------------------------------
 function [x,y,z] = katfilename(noplot)
 Matlab function to plot Finesse output data
 Usage:
 [x,y,z] = katfilename    : plots and returns the data
 [x,y,z] = katfilename(1) : just returns the data
           katfilename    : just plots the data
 Created automatically Wed Jun 11 11:34:17 2008
 by Finesse 0.99.8 (3200), 08.06.2008
------------------------------------------------------------------
```

This explains the three different possibilities to call the function and either load the data, plot the data or do both.

2.7 *.py—the Python script files

Similarly, these files are Python scripts that include the necessary commands to plot the FINESSE output with Python, using matplotlib.

This top comment block shows the usage:

```
"""----------------------------------------------------------------
  Python file for plotting Finesse ouput ttt.out
  created automatically Thu Apr 25 00:40:52 2013

  Run from command line as: python ttt.py
  Load from python script as: import ttt
  And then use:
  ttt.run() for plotting only
  x,y=ttt.run() for plotting and loading the data
  x,y=ttt.run(1) for only loading the data
----------------------------------------------------------------"""
```

Chapter 3

Mathematical description of light beams and optical components

3.1 Introduction

The following sections provide information about how the various aspects of an interferometer simulation are coded within the FINESSE source code. The analysis of optical systems described here is based on the principle of superposition of light fields: a laser beam can be described as the sum of different light fields. The possible degrees of freedom are:

- frequency,
- geometrical shape and position,
- polarisation.

In the analysis of interferometric gravitational wave detectors, the amplitudes and frequencies of light fields are of principal interest. The polarisation is neglected in the analysis given here, but the formalism can in principle be easily extended to include polarisation also.

This chapter describes the mathematical formalism based on plane waves only. In Chapter 4 the formalism with respect to Hermite-Gauss modes will be given; it is a straightforward extension of the plane wave analysis and makes use of the methods described here.

3.1.1 Static response and frequency response

The optical system shall be modelled by a set of linear equations that describes the light field amplitudes in a steady state. When a vector of input fields is provided, the set of linear equations can be mathematically solved by computing a *solution vector* that holds the field amplitudes at every component in the optical system.

The analysis provides information about the light field amplitudes as a function of the parameters of the optical system. Two classes of calculations can be performed:

a) **Static response**: Computing the light field amplitudes as a function of a quasi-static change of one or more parameters of the optical components. For example, the amplitude of a light field leaving an interferometer as a function of a change in an optical path length. The settling time of the optical system can usually be estimated using the optical parameters. Parameter changes that are negligible during the settling time can be assumed to be quasi-static. In a well-designed optical system many parameter changes can be treated as quasi-static so that the static response can be used to compute, for example, the (open-loop) error signal of the optical system's control loop.
b) **Frequency response**: In general, the frequency response describes the behaviour of an output signal as a function of the frequency of a fixed input signal. In other words, it represents a transfer function; in this context, a transfer function of an optical system. The input signal is commonly the modulation of light fields at some point in the interferometer. The frequency response allows computation of the optical transfer functions as, for example, required for designing control loops.

3.1.2 Transfer functions and error signals

Two common tasks for interferometer analysis are the computations of error signals and optical transfer functions. Both are important for the design of servo loops to control the interferometer. In an interferometer, several degrees of freedom for the optical components exist (e.g. positions, alignment angles) and active stabilisation is necessary to enhance the sensitivity.

An error signal is the output of a sensor (or in general a measurable signal) as a function of one degree of freedom (of the interferometer). The transfer function now gives the frequency-dependent coupling of a signal that is present in that particular 'degree of freedom' into the error signal.

Transfer functions can be used to compute the coupling of noise in the interferometer and thus to estimate the sensitivity. The following sections give an introduction into the computation of error signals and transfer functions with FINESSE.

Modulation-demodulation methods

Several standard techniques exist to generate error signals for controlling an interferometer. Many of them use modulation-demodulation schemes in which at some point inside the optical setup a light field is modulated (in phase or amplitude) at a fixed frequency. To derive error signals, the output of a photodetector is then demodulated (using a mixer) at that frequency. Modulation-demodulation is a well known technique which is commonly used for the transmission of low frequency signals (e.g. radio transmission). It has the advantage of shifting low frequency signals to higher frequencies. Typically, many noise contributions are frequency dependent such that the noise decreases at higher

frequencies. Therefore, the signal-to-noise ratio can be enhanced in many cases using modulation-demodulation.

Error signals

In general, an error signal is an output of any kind of detector that is suitable for stabilising a certain parameter p with a servo loop. Therefore, the error signal must be a function of the parameter p. In most cases it is preferable to have a bipolar signal with a zero crossing at the operating point p_0. The slope of the error signal at the operating point is a measure of the 'gain' of the sensor (which in general is a combination of optics and electronics).

Transfer functions

Transfer functions describe the propagation of a periodic signal through a *plant* and are usually given as frequency plots of amplitude and phase. A transfer function describes the *linear* coupling of signals inside a system. This means a transfer function is independent of the actual signal size. For small signals or small deviations, most systems can be linearised and correctly described by transfer functions.

Experimentally, network analysers are commonly used to measure a transfer function: One connects a periodic signal (the *source*) to an actuator of the plant (which is to be analysed) and to an input of the analyser. A signal from a sensor that monitors a certain parameter of the plant is connected to the second analyser input. By mixing the source with the sensor signal the analyser can determine the amplitude and phase of the input signal with respect to the source (amplitude equals one and the phase equals zero when both signals are identical).

In FINESSE transfer with the limitation that all *plants* are—of course—optical systems. The command:

```
fsig name component [type] f phase [amp]
```

specifies the source signal. One has to set a frequency and a phase and can optionally set an amplitude and the type of the signal. Giving an amplitude or phase makes sense only if the frequency is applied to more than one component and the *relative* driving phases and amplitudes are of interest. **A signal can be added to the following components: mirror, beam splitter, space, input, and modulator**. All of these can add a modulation to a light field, i.e. they can act dynamically on the light field amplitudes. In all cases the modulation is only applied to laser fields or RF modulation sidebands (i.e. those which are generated by a modulator component). The practical reason for this restriction is the difficulty of avoiding endless loops when signal sidebands are generated around signal sidebands. From the physics point of view the restriction also makes sense since transfer functions can be calculated with infinitesimally small signals (i.e. perturbations of order ϵ) so that all terms of the order ϵ^2 can be omitted. In addition

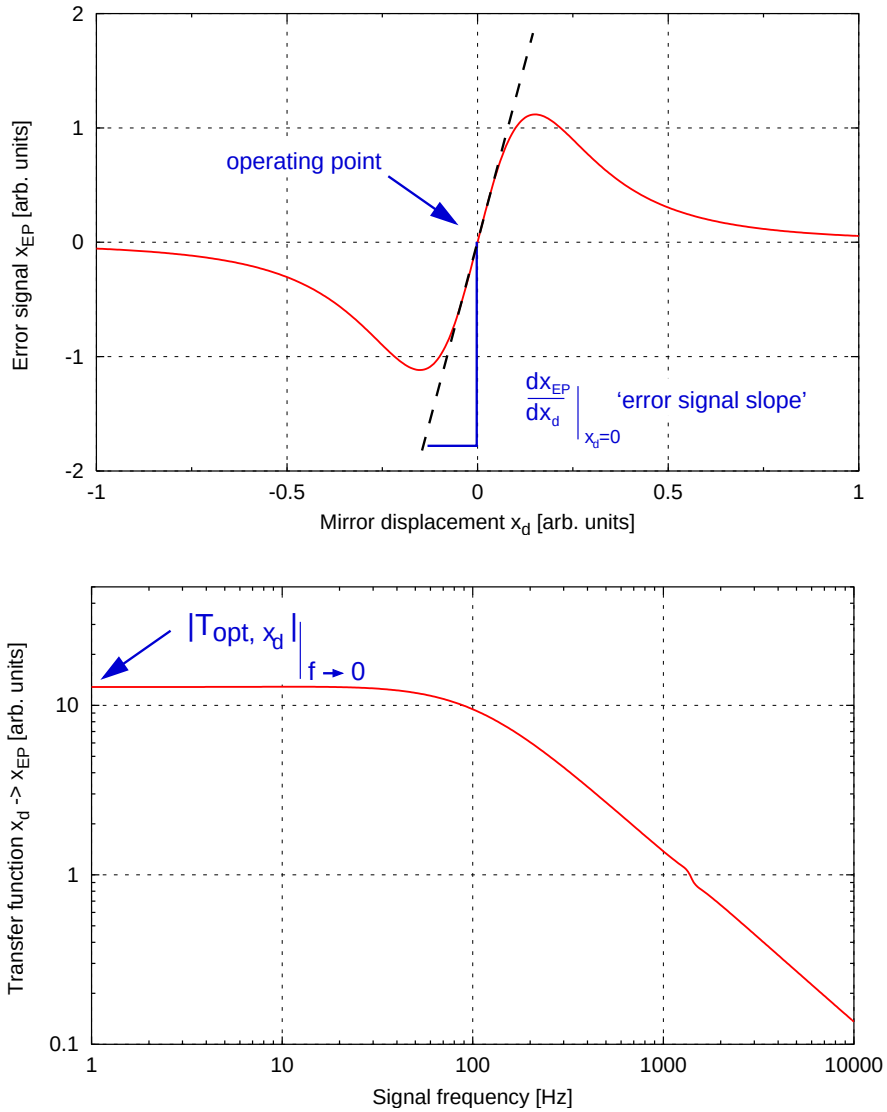


Figure 3.1: *Example of an error signal: the top graph shows the electronic interferometer output signal as a function of mirror displacement. The operating point is given as the zero crossing, and the* error-signal slope *is defined as the slope at the operating point. The bottom graph shows the magnitude of the transfer function* mirror displacement → error signal. *The slope of the error signal (upper graph) is equal to the low frequency limit of the transfer function magnitude (see Equation 3.3).*

to supply a constant phase and amplitude as above, you can also define a transfer function as the scaling as a function of frequency

```
fsig name component [type] f transfer_function
```

On defining transfer functions see section 3.1.2. The signal frequency is also has a special

variable associated with it that can be used in functions and detector commands: `$fs` for positive and `$mfs` for the negative signal frequency.

In FINESSE, modulation at the signal frequency is realised by adding two signal sidebands to the light field. Applying the signal to the various components results in different amplitudes and phases of these sidebands. The exact numbers for each possible component are given in Section 3.4.5. The detection of the signal for creating a transfer function is included in the photodetector components `pd` (see below).

Figure 3.1 shows an example of an error signal and its corresponding transfer function. The operating point will be at:

$$x_{\mathrm{d}} = 0 \qquad \text{and} \qquad x_{\mathrm{EP}}(x_{\mathrm{d}} = 0) = 0 \tag{3.1}$$

The optical transfer function $T_{\mathrm{opt,x_d}}$ with respect to this error signal is defined by:

$$\widetilde{x}_{\mathrm{EP}}(f) = T_{\mathrm{opt,x_d}} T_{\mathrm{det}} \widetilde{x}_d(f), \tag{3.2}$$

with T_{det} as the transfer function of the sensor. In the following, T_{det} is assumed to be unity. At the zero crossing the slope of the error signal represents the magnitude of the transfer function for low frequencies:

$$\left|\frac{dx_{\mathrm{EP}}}{dx_{\mathrm{d}}}\right|_{x_{\mathrm{d}}=0} = |T_{\mathrm{opt,x_d}}|_{f\to 0} \tag{3.3}$$

The quantity above will be called the *error-signal slope* in the following text. It is proportional to the *optical gain* $|T_{\mathrm{opt,x_d}}|$, which describes the amplification of the gravitational wave signal by the optical instrument.

User-defined transfer functions

Transfer functions are also used in FINESSE for radiation pressure computations and linking of components. Such transfer functions describe a frequency dependent signal; the signal can represent, for example, how a force applied to a component results in a motion, or the change of an electronic signal between a photodiode output and an mirror actuator.

The transfer function commands in FINESSE are generic and do not express the units or meaning of the signal transformation. The `tf` command is used to define a transfer function as a sum of *poles* and *zeros*,

```
tf name factor phase [p/z f1 Q1 [p/z f2 Q2] [p/z f3 Q3 ...]]
```

`factor` and `phase` are the overall gain and phase of the transfer function.

The number of poles and zeros is not limited, for example you can specify none of either to get a flat transfer function at all frequencies. the option `p/z` is exclusive-or, `p` being

to define a pole and `z` a zero. You then state the frequency of the pole or zero and the quality factor of the resonance: $0 < Q < 0.5$ is over-damped, $Q = 0.5$ critically damped and $Q > 0.5$ is under-damped.

For example, if for radiation pressure modelling we want to specify the transfer function from force applied to longitudinal motion for a single pendulum with resonance at 2 Hz and a quality factor $Q = 10^6$, you would use the command:

```
tf sus 1 0 p 2 1M
```

There is also the option to specify complex values directly for the poles and zeros:

```
tf2 name factor phase {p1,p2,...} {z1,z2,...}
```

Instead of supplying the resonance frequency and quality factor we state a string of complex numbers for the poles and zeros. Each pole and zero should always be a conjugate pair. The curly braces should always be present even if no pole or zeros are added, for example

```
tf2 sus 1 0 {} {}
```

would be a valid command. Or we can define some values

```
tf2 sus 1 0 {1+100i,1-100i,0+10i+0-10i} {10+200i,10-200i,2+0i,2-0i}
```

Note that we have defined conjugate pairs, there are no spaces, we always state the real and imaginary parts even if they are numerically 0 and that the letter i always comes after the numerical value of the imaginary part. The parsing of these complex numbers is fragile and these rules mut be followed carefully.

3.1.3 The interferometer matrix

The task for FINESSE is to compute the coupling of light field amplitudes inside a given interferometer. FINESSE assumes the following simplifications:

- the interferometer can be described via linear coupling of the light field amplitudes,
- there is no polarisation of the light, nor polarising components,
- the frequency of a given light field is never changed, in particular frequency shifting is not possible.

With these simplifications all interactions at optical components can be described by a simple set of linear equations. For a given number of input fields this set of equations can be 'solved' (either numerically or analytically) and the output fields can be computed. FINESSE first creates local matrices with the local coupling coefficients for every optical component. Next, the full interferometer matrix is compiled from these 'local' coupling matrices. The full interferometer matrix then transforms a vector with all local fields (the 'solution' vector) into a vector that contains non-zero entries for the input light fields in

all interferometer inputs. The latter is called the 'right hand side' (RHS) vector.

$$\begin{pmatrix} \text{interferometer} \\ \text{matrix} \end{pmatrix} \times \begin{pmatrix} \vec{x}_{\text{sol}} \\ \\ \end{pmatrix} = \begin{pmatrix} \vec{x}_{\text{RHS}} \\ \\ \end{pmatrix} \tag{3.4}$$

The number of rows (the matrix is of the type $n \times n$) is determined by the number of distinct light field amplitudes inside the interferometer. If, for example, we consider only one frequency component and one geometrical mode, exactly two light fields are present at every node and the number of rows is two times the number of nodes.

FINESSE makes a clear distinction between types of optical fields: carrier fields, which are those created by a `laser` component and their sidebands create by `modulator` components; signal fields, which are always considered much smaller in magnitude than any carrier field; and quantum sideband fields, which is descibed in more detail in later chapters. Each field type is solved with a separate matrix, the carrier matrix solves all carrier fields, the signal matrix solves all the signal fields, and the quantum sideband matrix solves all the quantum fields; these will be refered to at various points in later chapters. The carrier matrix is always solved first and the values computed are used to create sources for the signal fields, thus it is always assumed that signal fields in no way affect the carrier fields.

The RHS vector consists mostly of zeros since usually there are only a few distinct sources of light in an interferometer. These sources are 'lasers' for the carrier matrix and 'signals' that are applied to various optical components for computing transfer functions in the signal matrix. The 'signals' shift light power from carrier light fields at a specified frequency, called the *signal frequency* typically denoted as f_s, therefore they can be treated as light sources (in general as devices that can create or destroy light power at a given frequency).

Naturally, the entries in the matrix and in the RHS vector change during a simulation. In fact, the coefficients of the matrix are updated every time a parameter has been changed. Then an RHS vector is set up and the system of linear equations is solved numerically. The solution vector is computed and thus the field amplitudes at all nodes and for all frequencies inside the interferometer.

3.2 Conventions and concepts

This section presents an overview of the conventions and definitions that are used in FINESSE. Several methods for describing the same physics are commonly used. Therefore, the knowledge of the definitions used by FINESSE is essential for understanding the syntax of the input files, the results of the computation and, in some cases, the descriptions in this manual.

3.2.1 Nodes and components

The interferometer has to be specified as a group of components connected by 'nodes'. For example, a two mirror Fabry-Perot cavity as in Figure 3.2 could be:

- mirror one (`m1`) with nodes `n1` and `n2`
- free space (`s`) with nodes `n2` and `n3`
- mirror two (`m2`) with nodes `n3` and `n4`

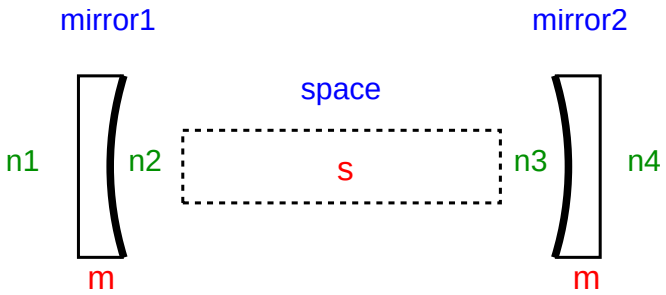


Figure 3.2: *A simple Fabry-Perot cavity shown as components and nodes.*

Node `n2` connects mirror `mirror1` to space `s` and node `n3` connects space `s` to mirror `mirror2`. The nodes `n1` and `n4` are now the input and output nodes of the cavity. The program calculates the light fields at all nodes. As opposed to reality, you can put a detector at every node without disturbing the interferometer. **Two light fields are present at every node: one in each direction of propagation**, e.g. in the example above at node `n2`, one light field approaching the mirror `mirror1` from the right and one leaving the mirror `mirror1` to the right. If a detector is located at node `n2`, only **one** of these two fields is detected; you have to know (or specify) which. If the node where you put a detector has only one connection (like `n4` above), the non-empty light field (i.e. coming from the mirror `mirror2`) is chosen automatically. If the node is inside the system (e.g. `n2`) **the beam going into a space coming from a different component is usually detected**. If there are no spaces in between, the following rules apply in the given order:

- if a mirror is connected to the node, the beam coming *from* the mirror is detected,
- if a beam splitter is connected to the node, the beam coming *from* the beam splitter is detected,
- if a modulator is connected to the node, the beam coming *from* the modulator is detected,
- if two components of the same type (i.e. two mirrors) are connected to the node, the beam coming from the first specified component (in the input file) is detected.

The rules above state which beam is detected by default. Of course, you can specify that the respective other beam should be detected (see syntax reference for '`ad`' or '`pd`' in Appendix G).

3.2.2 Mirrors and beam splitters

The main components of interferometers in FINESSE are mirrors and beam splitters. Following the implementation of the respective numerical methods used in FINESSE the mirrors and beam splitters are defined in a slightly counter-intuitive way:

- mirrors (`m`) are defined as single optical surfaces with two nodes
- beam splitters (`bs`) are defined as single optical surfaces with four nodes

In other words, a mirror always retro-reflects a beam into the incoming node while a beam splitter separates the reflected beam from the incoming beam.

The counter-intuitive part in this is that when a real beam splitter (i.e. a partial reflective surface) is used under normal incidence we would still call it a beam splitter while in FINESSE it has to be modeled as mirror. On the other hand if we employ a mirror as a turning mirror, a mirror of $R = 1$ with an angle of incidence of 45 degrees, this must be modelled as a beam splitter in FINESSE.

In short the terms mirror and beam splitter do *not* refer to the reflectance or transmission of the optical surface nor how the modelled optic is used in practise, the difference is solely defined by the angle of incidence of the incoming beam.

Please note also that even though many diagrams in this manual depict mirrors and beam splitters as components with two surfaces the actual components `m` and `bs` represent *single* optical surfaces. Such surfaces of course do not resemble physical objects as such. However, they can be used to model simplified interferometer layouts. In more detailed optical layouts it is often wise to use more realistic models for the optical components: a 'real' mirror or beam splitter would consist of two optical surfaces with a substrate in between. To model this one needs to employ a number of basic FINESSE components. A mirror then can be modeled as:

```
m Mfront ...nM1 nMi1
s Msubstrate ...nMi1 nMi2
m Mback ...nMi2 nM2
```

A beam splitter is more complex because the outgoing beams would pass the back surface at different location. FINESSE cannot handle this directly, instead one has to consider the two locations at the back surface as two independent components (this also makes sense in practice since the surface properties for the two locations are not necessarily equal). Thus a beam splitter can be modeled as:

```
bs BSfront ...nBS1 nBS2 nBSi1 nBSi3
s BSsubstrate1 ...nBSi1 nBSi2
s BSsubstrate2 ...nBSi3 nBSi4
bs BSback1 ...nBSi2 nBS3
bs BSback2 ...nBSi4 nBS4
```

3.3 Frequencies and wavelengths

FINESSE distinguishes between three types of light fields:

- laser light (input light),
- modulation sidebands (generated by the component 'modulator': `mod`), and
- signal sidebands (generated by the command 'signal frequency': `fsig`).

Throughout this manual, different representations for the frequencies of light fields are used: wavelength (λ), frequency (f), or angular frequency (ω). In FINESSE it is simpler: in the file 'kat.ini' the default laser frequency is defined via a wavelength λ_0. All other frequencies must be given as frequency offsets (**not angular frequency**) to that reference. The terms 'carrier' frequency or 'carrier' light are used in this manual to refer to a light field that is subject to some kind of modulation, either by a modulator or a signal. The modulation will create 'sidebands' around the 'carrier'. Laser light fields can be carriers. But also modulation sidebands created by the component `mod` can serve as carrier fields to subsequent modulations.

A modulator can be used as a phase or amplitude modulator. In addition, a modulator can be used in the *single sideband mode* so that only one modulation sideband is added to the laser field. Signal frequencies perform a modulation on *input fields* and *modulation sidebands* from a modulator but **not** on other signal sidebands.

3.3.1 Phase change on reflection and transmission

When a light field passes a beam splitter, a phase jump in either the reflected, transmitted, or both fields is required for energy conservation; the actual phase change for the different fields depends on the type of beam splitter (see [Rüdiger] and [Heinzel]). In practice, the absolute phase of the light field at a beam splitter is of little interest so to calculate interferometer signals one can choose a convenient implementation for the relative phase. Throughout this work, the following convention is used: mirrors and beam splitters are assumed to be symmetric (not in how they split the light power but with respect to the phase change) and the phase is not changed upon reflection; instead, the phase changes by $\pi/2$ at every transmission.

Be aware that this is directly connected to the resonance condition in the simulation: if, for example, a single surface with power transmittance $T = 1$ is inserted into a simple cavity, the extra phase change by the transmission will change the resonance condition to its opposite. Inserting a 'real' component with two surfaces, however, does not show this effect.

3.3.2 Lengths and tunings

The interferometric gravitational wave detectors typically use three different types of light fields: the laser with a frequency of $f \approx 2.8 \cdot 10^{14}$ Hz, modulation sidebands used for

interferometer control with frequencies (offsets to the laser frequency) in the MHz range, for example $f \approx 30 \cdot 10^6$ Hz, and the signal sidebands at frequencies of 10 Hz to 1000 Hz[1].

The resonance condition inside the cavities and the operating point of the interferometer depend on the optical path lengths modulo the laser wavelength, i.e. for the light of a Nd:YAG laser length differences of less than 1 μm are of interest, not the absolute length. The propagation of the sideband fields depends on the much larger wavelength of the (offset) frequencies of these fields and thus often on absolute lengths. Therefore, it is convenient to split distances D between optical components into two parameters [Heinzel]: one is the macroscopic 'length' L defined as that multiple of the default wavelength λ_0 yielding the smallest difference to D. The second parameter is the microscopic *tuning* that is defined as the remaining difference between L and D. This tuning is usually given as a phase ϕ (in radian) with 2π referring to one wavelength[2]. In FINESSE tunings are entered and printed in degrees, so that a tuning of $\phi = 360$ degrees refers to a change in the position of the component by one wavelength (λ_0).

This convention provides two parameters that can describe distances with a markedly improved numerical accuracy. In addition, this definition often allows simplification of the algebraic notation of interferometer signals.

In the following, the propagation through free space is defined as a propagation over a macroscopic length L, i.e. a free space is always 'resonant', i.e. a multiple of λ_0. The microscopic tuning appears as a parameter of mirrors and beam splitters. It refers to a microscopic displacement perpendicular to the surface of the component. If, for example, a cavity is to be resonant to the laser light, the tunings of the mirrors have to be the same whereas the length of the space in between can be arbitrary.

Note that if you change the frequency of the input lasers the spaces are still resonant to the default wavelength λ_0 (as given in 'kat.ini') and not to the wavelength of the input light.

3.4 The plane-wave approximation

In many simulations the shape of the light beams or, in general, the geometric properties of a beam transverse to the optical axis are not of interest. In that case one can discard this information and restrict the model to the field on the optical axis. This is equivalent to a model where all light fields are plane waves traveling along one optical axis. This is the standard mode of FINESSE and is called *plane-wave approximation* in the following.

In the plane-wave approximation all light fields are described in one dimension. All beams and optical components are assumed to be centered on the optical axis and of infinite size.

[1] The signal sidebands are sometimes also called *audio sidebands* because of their frequency range.

[2] Note that in other publications the tuning or equivalent microscopic displacements are sometimes defined via an optical path length difference and then often 2π is used to refer to the change of the optical path length of one wavelength which, for example, if the reflection at a mirror is described, corresponds to a change of the mirror's position of $\lambda_0/2$.

Using plane waves, it is very simple to compute interferometer signals depending on the phase and frequency of the light, and of the longitudinal degrees of freedom. Furthermore it can be easily extended to include other degrees of freedom, such as polarisation or transverse beam shapes.

This section introduces the plane-wave approximation as used in FINESSE by default. It also presents the basis for the Hermite-Gauss extension given in Section 4.

3.4.1 Description of light fields

A laser beam is usually described by the electric component of its electromagnetic field:

$$\vec{E}(t,\vec{x}) \;=\; \vec{E}_0 \cos\left(\omega\, t - \vec{k}\vec{x}\right). \tag{3.5}$$

In the following calculations, only the scalar expression for a fixed point in space is used. The calculations can be simplified by using the full complex expression instead of the cosine:

$$E(t) \;=\; E_0 \,\exp\left(\mathrm{i}\;(\omega\, t + \varphi)\right) \;= a\,\exp\left(\mathrm{i}\,\omega\, t\right), \tag{3.6}$$

where $a = E_0\,\exp\left(\mathrm{i}\,\varphi\right)$. The real field at that point in space can then be calculated as:

$$\vec{E}(t) = \mathrm{Re}\left\{E(t)\right\}\cdot\vec{e}_{\mathrm{pol}}, \tag{3.7}$$

with $\vec{e}_{\mathrm{pol}}$ as the unit vector in the direction of polarisation.

Each light field is then described by the complex amplitude a and the angular frequency ω. Instead of ω, also the frequency $f = \omega/2\pi$ or the wavelength $\lambda = 2\pi c/\omega$ can be used to specify the light field. It is often convenient to define one *default frequency* (also called the *default laser frequency*) f_0 as a reference and describe all other light fields by the offset Δf to that frequency. In the following, some functions and coefficients are defined using f_0, ω_0, or λ_0 referring to a previously defined default frequency. The setting of the default frequency is arbitrary, it merely defines a reference for frequency offsets and does not influence the results.

The electric component of electromagnetic radiation is given in Volt per meter. The light power computes as:

$$P = \frac{\epsilon_0 c}{2}\; EE^*, \tag{3.8}$$

with ϵ_0 the electric permeability of vacuum and c the speed of light. However, for more intuitive results the light fields can be given in converted units, so that the light power can be computed as the square of the light field amplitudes. Unless otherwise noted, throughout this work the unit of light field amplitudes is the square root of Watt. Thus, the power computes simply as:

$$P = EE^*. \tag{3.9}$$

The parameter 'epsilon_c' in the init file 'kat.ini' can be used to set the value of $\epsilon_0 \cdot c/2$ (the default is $\epsilon_0 \cdot c/2 = 1$).

3.4.2 Photodetectors and mixers

In plane-wave mode FINESSE offers two methods for detecting light in an interferometer, amplitude detectors and photodetectors. An amplitude detector (`ad`) detects **only** the light amplitude at the given frequency even if other light fields are present. An amplitude detector is a virtual device.

A photodetector (`pd`) does not only detect light at the given frequency, but also beat signals at that frequency. For example, at DC the photodetector detects the full DC power of all present light fields. This 'photodetector' refers to real photodetectors except for the fact that it does not destroy (or change in any sense) the light field.

The photodetectors can perform a demodulation of the detected signal (light power). In reality this would be done by a mixer. In FINESSE photodetectors can be specified with up to 5 mixer frequencies and phases: when a mixer frequency (and phase) is given, the signal is demodulated at this frequency. When more frequencies are specified, the signal is demodulated at these frequencies sequentially. Please note that FINESSE does not simulate a mixer: in the frequency domain the demodulation can be achieved by simply selecting only amplitudes at the modulation frequency when computing the output of a photodetector (see Section 3.4.3).

A real mixer always demodulates the signal with a certain demodulation phase. The output is then a real number which represents the amplitude of the signal at the specified frequency and phase. More information can be obtained if two mixers are used with different demodulation phases. Using two mixers that demodulate the same signal at the same frequency but with a difference in the demodulation phase of $\pi/2$ the amplitude and phase of the signal at the specified frequency can be reconstructed. This is used in network analysers to measure transfer functions.

In FINESSE the demodulation automatically preserves the phase of the signal anyway. If a demodulation phase is specified, the complex amplitude is projected onto that phase and thus converted to a real number. On the other hand, a network analyser can be simulated by simply leaving out the last step: if the demodulation phase for the last specified frequency is omitted, FINESSE keeps the full complex amplitude. This feature is (as in network analysers) commonly used for computing transfer functions.

3.4.3 Modulation of light fields

In principle, all parameters of a light field can be modulated. This section describes the modulation of the amplitude, phase and frequency of the light.

Any sinusoidal modulation of amplitude or phase generates new field components that are shifted in frequency with respect to the initial field. Basically, light power is shifted from one frequency component, the *carrier*, to several others, the *sidebands*. The relative amplitudes and phases of these sidebands differ for different types of modulation and different modulation strengths.

Phase modulation

Phase modulation can create a large number of sidebands. The amount of sidebands with noticeable power depends on the modulation strength (or depths) given by the *modulation index* m.

Assuming an input field:

$$E_{\text{in}} = E_0 \exp(\mathrm{i}\,\omega_0\,t), \tag{3.10}$$

a sinusoidal phase modulation of the field can be described as:

$$E = E_0 \exp\Big(\mathrm{i}\,(\omega_0\,t + m\cos(\omega_{\text{m}}\,t))\Big). \tag{3.11}$$

This equation can be expanded using the Bessel functions $J_k(m)$ to:

$$E = E_0 \exp(\mathrm{i}\,\omega_0\,t) \sum_{k=-\infty}^{\infty} \mathrm{i}^{\,k}\, J_k(m) \exp(\mathrm{i}\,k\omega_{\text{m}}\,t). \tag{3.12}$$

The field for $k = 0$, oscillating with the frequency of the input field ω_0, represents the carrier. The sidebands can be divided into *upper* ($k > 0$) and *lower* ($k < 0$) sidebands. These sidebands are light fields that have been shifted in frequency by $k\,\omega_{\text{m}}$. The upper and lower sidebands with the same absolute value of k are called a pair of sidebands of order k.

Equation 3.12 shows that the carrier is surrounded by an infinite number of sidebands. However, the Bessel functions decrease for large k, so for small modulation indices ($m < 1$), the Bessel functions can be approximated by:

$$J_k(m) = \frac{1}{k!}\left(\frac{m}{2}\right)^k + O\left(m^{k+2}\right). \tag{3.13}$$

In which case, only a few sidebands have to be taken into account. For $m \ll 1$ we can write:

$$\begin{aligned} E = {} & E_0 \exp(\mathrm{i}\,\omega_0\,t) \\ & \times\Big(J_0(m) - \mathrm{i}\,J_{-1}(m) \exp(-\mathrm{i}\,\omega_{\text{m}}\,t) + \mathrm{i}\,J_1(m) \exp(\mathrm{i}\,\omega_{\text{m}}\,t)\Big), \end{aligned} \tag{3.14}$$

and with

$$J_{-k}(m) = (-1)^k J_k(m), \tag{3.15}$$

we obtain:

$$E = E_0 \exp(\mathrm{i}\,\omega_0\,t) \left(1 + \mathrm{i}\,\frac{m}{2}\Big(\exp(-\mathrm{i}\,\omega_{\text{m}}\,t) + \exp(\mathrm{i}\,\omega_{\text{m}}\,t)\Big)\right), \tag{3.16}$$

as the first-order approximation in m.

When the modulator functions as a phase modulator, then the *order* of sidebands can be given. For example:

```
mod eom1 10M 0.6 2 pm node1 node2
```

applies a cosine phase modulation at 10 MHz with a modulation index of $m = 0.6$ and order 2, i.e. 4 sidebands are added to the laser field.

The given number for *order* in the modulator command simply specifies the highest order of Bessel function which is to be used in the sum in Equation 3.12, i.e. the program code uses the equation:

$$E = E_0 \exp(\mathrm{i}\,\omega_0\,t) \sum_{k=-order}^{order} i^k\, J_k(m) \exp(\mathrm{i}\,k\omega_m\,t), \tag{3.17}$$

Frequency modulation

For small modulation indices phase modulation and frequency modulation can be understood as different descriptions of the same effect [Heinzel]. With the frequency defined as $f = d\varphi/dt$ a sinusoidal frequency modulation can be written as:

$$E = E_0 \exp\left(\mathrm{i}\left(\omega_0\,t + \frac{\Delta\omega}{\omega_\mathrm{m}}\cos(\omega_\mathrm{m}\,t)\right)\right), \tag{3.18}$$

with $\Delta\omega$ as the frequency swing (how *far* the frequency is shifted by the modulation) and ω_m the modulation frequency (how *fast* the frequency is shifted). The modulation index is defined as:

$$m = \frac{\Delta\omega}{\omega_\mathrm{m}}. \tag{3.19}$$

Amplitude modulation

In contrast to phase modulation, (sinusoidal) amplitude modulation always generates exactly two sidebands. Furthermore, a natural maximum modulation index exists: the modulation index is defined to be one ($m = 1$) when the amplitude is modulated between zero and the amplitude of the unmodulated field.

If the amplitude modulation is performed by an active element, for example by modulating the current of a laser diode, the following equation can be used to describe the output field:

$$\begin{aligned} E &= E_0 \exp(\mathrm{i}\,\omega_0\,t)\left(1 + m\cos(\omega_\mathrm{m}\,t)\right) \\ &= E_0 \exp(\mathrm{i}\,\omega_0\,t)\left(1 + \tfrac{m}{2}\exp(\mathrm{i}\,\omega_\mathrm{m}\,t) + \tfrac{m}{2}\exp(-\mathrm{i}\,\omega_\mathrm{m}\,t)\right). \end{aligned} \tag{3.20}$$

However, passive amplitude modulators (like acousto-optic modulators or electro-optic modulators with polarisers) can only reduce the amplitude. In these cases, the following

equation is more useful:

$$\begin{aligned} E &= E_0 \ \exp(\mathrm{i}\,\omega_0\,t) \ \left(1 - \tfrac{m}{2}\left(1 - \cos(\omega_\mathrm{m}\,t)\right)\right) \\ &= E_0 \ \exp(\mathrm{i}\,\omega_0\,t) \ \left(1 - \tfrac{m}{2} + \tfrac{m}{4} \ \exp(\mathrm{i}\,\omega_\mathrm{m}\,t) + \tfrac{m}{4} \ \exp(-\mathrm{i}\,\omega_\mathrm{m}\,t)\right). \end{aligned} \tag{3.21}$$

Single sideband

The modulator components in FINESSE can be switched to a *single sideband mode* where only one sideband is added to the input light. This sideband can be either identical to one phase modulation sideband or one amplitude modulation sideband (see above). The modulation index is used as usual, so that a single sideband created with modulation index m has the same amplitude as, for example, the upper sideband in an ordinary phase modulation (order=1) with modulation index m. Therefore, the amplitude of the input field remains larger in the single sideband case. If A_0 is the amplitude of the input light before modulation and A_m is the amplitude of the carrier light after a normal modulation, the amplitude of the carrier after a single sideband modulation is:

$$A_{ssb} = A_0 - \frac{A_0 - A_m}{2}. \tag{3.22}$$

Oscillator phase noise

The oscillator phase noise (or modulator phase noise) can give some information about the performance of a modulation scheme in connection with a certain interferometer configuration. The term *phase noise* describes the change of the phase of the modulation frequency. In Equation 3.11 the phase of the modulation frequency was supposed to be zero and is not given explicitly. In general, the modulated light has to be written with a phase term:

$$E = E_0 \ \exp(\mathrm{i}\ (\omega_0\,t + m\cos(\omega_m\,t + \varphi_m(t))))\,. \tag{3.23}$$

Using Equation 3.17 phase noise can be expressed like this:

$$E = E_0 \ e^{\mathrm{i}\,\omega_0\,t} \sum_{k=-order}^{order} i^k \ J_k(m) \ e^{\mathrm{i}\,k(\omega_m\,t+\varphi_m(t))}. \tag{3.24}$$

To investigate the coupling of $\varphi_m(t)$ into the output signal, we apply a cosine modulation at the signal frequency (ω_noise):

$$\varphi_m(t) = m_2 \ \cos(\omega_\mathrm{noise}\,t), \tag{3.25}$$

which results in the following field:

$$E = E_0 \ e^{\mathrm{i}\,\omega_0\,t} \sum_{k=-order}^{order} i^k \ J_k(m) \ e^{\mathrm{i}\,k\omega_m\,t} \sum_{l=-\infty}^{\infty} i^l \ J_l(k\ m_2) \ e^{\mathrm{i}\,l\omega_\mathrm{noise}\,t}. \tag{3.26}$$

The extra modulation of φ_m thus adds extra sidebands to the light (which will be called 'audio sidebands' in the following since in most cases the interesting signal frequencies are from DC to some kHz whereas the phase modulation frequencies are very often in the MHz regime). The audio sidebands are generated around each phase modulation sideband. We are interested in the coupling of the audio sidebands into the interferometer output because these sidebands will generate a false signal and therefore limit the sensitivity of the interferometer. For a computation of a transfer function, the amplitude of the signal sidebands (here: modulation index of audio sidebands) is assumed to be very small so that only the terms for $l = -1, 0, 1$ in the second sum in Equation 3.26 have to be taken into account and the Bessel functions can be simplified to:

$$\begin{aligned} E \;=\; E_0 \;\; & e^{\mathrm{i}\,\omega_0\,t} \;\textstyle\sum_{k=-order}^{order} i^{\,k}\; J_k(m)\; e^{\mathrm{i}\,k\omega_m\,t} \\ & \times \left(1 + \mathrm{i}\,\tfrac{k\;m_2}{2}\; e^{-i\omega_{\mathrm{noise}}\,t} \;+ \mathrm{i}\,\tfrac{k\;m_2}{2}\; e^{\mathrm{i}\,\omega_{\mathrm{noise}}\,t} + O((km_2)^2)\right). \end{aligned} \tag{3.27}$$

FINESSE automatically generates the above signal sidebands for oscillator phase noise when the command `fsig` (see Section 3.1.2) is used with a modulator as component. For example,

```
fsig signal1 eom1 10k 0
```

adds audio sidebands ($\omega_{\mathrm{noise}} = 2\pi\,10\,\mathrm{kHz}$) to the modulation sidebands (which are generated by `eom1`).

Oscillator amplitude noise[3]

Oscillator amplitude noise has not yet been implemented in FINESSE. This section describes preparatory work towards a future implementation.

In order to derive the coupling between sidebands we start with a phase modulated light field,

$$E = E_0 \exp(i(\omega_0 t + m\cos(\omega_m t))) \tag{3.28}$$

and replace the modulation index of the phase modulation with one that is amplitude modulated with amplitude m_2 and frequency ω_{m_2},

$$m = m_1(1 + m_2\cos(\omega_{m_2} t)) \tag{3.29}$$

The light field now has form,

$$E = E_0 \exp(i(\omega_0 t + m_1\cos(\omega_{m_1} t) + m_1 m_2\cos(\omega_{m_1} t)\cos(\omega_{m_2} t))) \tag{3.30}$$

Using the identity,

$$\cos(A)\cos(B) = \frac{1}{2}(\cos(A+B) + \cos(A-B)) \tag{3.31}$$

[3] This section has been contributed by Joshua Smith

we can obtain,

$$\begin{aligned} E &= E_0 \exp(i\omega_0 t) \\ &\quad \cdot \exp(i m_1 \cos(\omega_{m_1} t)) \\ &\quad \cdot \exp(i \frac{m_1 m_2}{2} \cos((\omega_{m_1} + \omega_{m_2}) t) \\ &\quad \cdot \exp(i \frac{m_1 m_2}{2} \cos((\omega_{m_1} - \omega_{m_2}) t) \end{aligned} \tag{3.32}$$

This can be expanded into three sums of Bessel functions following Equation 3.12 and Equation 3.13

$$\begin{aligned} E &= E_0 \exp(i\omega_0 t) \\ &\quad \cdot \sum_{k=-\infty}^{\infty} i^k J_k(m_1) \exp(ik\omega_{m_1} t) \\ &\quad \cdot \sum_{l=-\infty}^{\infty} i^l J_l(m_{12}) \exp(il(\omega_{m_1} + \omega_{m_2}) t) \\ &\quad \cdot \sum_{n=-\infty}^{\infty} i^n J_n(m_{12}) \exp(in(\omega_{m_1} - \omega_{m_2}) t) \\ &= E_0 \exp(i\omega_0 t) \left(\sum_{k=-\infty}^{\infty} i^k J_k(m_1) \exp(ik\omega_{m_1} t) \right) \cdot C \end{aligned} \tag{3.33}$$

with

$$\begin{aligned} C &= \sum_{l=-\infty}^{\infty} i^l J_l(m_{12}) \exp(il(\omega_{m_1} + \omega_{m_2}) t) \\ &\quad \cdot \sum_{n=-\infty}^{\infty} i^n J_n(m_{12}) \exp(in(\omega_{m_1} - \omega_{m_2}) t) \end{aligned} \tag{3.34}$$

where $m_{12} = m_1 m_2 / 2$. As before we restrict this analysis to small modulation indices m_2 and only consider sidebands with $l = 0, \pm 1$ and $n = 0, \pm 1$. This yields

$$\begin{aligned} C &= \left(1 + i \frac{m_{12}}{2} \exp(-i(\omega_{m_1} + \omega_{m_2}) t) + i \frac{m_{12}}{2} \exp(i(\omega_{m_1} + \omega_{m_2}) t)\right) \\ &\quad \cdot \left(1 + i \frac{m_{12}}{2} \exp(-i(\omega_{m_1} - \omega_{m_2}) t) + i \frac{m_{12}}{2} \exp(i(\omega_{m_1} - \omega_{m_2}) t)\right) \\ &= 1 + i \frac{m_{12}}{2} \left(\exp(-i(\omega_{m_1} + \omega_{m_2}) t) + \exp(i(\omega_{m_1} + \omega_{m_2}) t)\right. \\ &\quad \left. + \exp(-i(\omega_{m_1} - \omega_{m_2}) t) + \exp(i(\omega_{m_1} - \omega_{m_2}) t)\right) + O(m_2^2) \\ &= 1 + i \frac{m_{12}}{2} \left(\exp(i\omega_{m_1} t) + \exp(-i\omega_{m_1} t)\right) \left(\exp(i\omega_{m_2} t) + \exp(-i\omega_{m_2} t)\right) + O(m_2^2) \end{aligned} \tag{3.35}$$

3.4.4 Coupling of light field amplitudes

Many optical systems can be described mathematically using linear coupling of light field amplitudes. Passive components, such as mirrors, beam splitters and lenses, can be described well by linear coupling coefficients. Active components, such as electro-optical modulators cannot be described so easily. Nevertheless, simplified versions of active components can often be included in a linear analysis.

The coupling of light field amplitudes at a simple (flat, symmetric, etc.) mirror under normal incidence can be described as follows: there are two input fields, In1 impinging on the mirror on the front surface and In2 on the back surface. Two output fields leave the mirror, Out1 and Out2. With the amplitude coefficients for reflectance and transmittance (r, t) the following equations can be composed:

$$\begin{aligned} \mathrm{Out1} &= r\ \mathrm{In1} + \mathrm{i}\,t\ \mathrm{In2}, \\ \mathrm{Out2} &= r\ \mathrm{In2} + \mathrm{i}\,t\ \mathrm{In1}. \end{aligned} \tag{3.36}$$

Possible loss is included in this description because the sum $r^2 + t^2$ may be less than one; see Section 3.3.1 about the convention for the phase change.

The above equations completely define this simplified optical component. Optical systems that consist of similar components can be described by a set of linear equations. Such a set of linear equations can easily be solved mathematically, and the solution describes the equilibrium of the optical system: given a set of input fields (as the 'right hand side' of the set of linear equations), the solution provides the resulting field amplitudes everywhere in the optical system. This method has proven to be very powerful for analysing optical systems. It can equally well be adapted to an algebraic analysis as to a numeric approach.

In the case of the plane-wave approximation, the light fields can be described by their complex amplitude and their angular frequency. The linear equations for each component can be written in the form of local coupling matrices in the format:

$$\begin{pmatrix} \mathrm{Out1} \\ \mathrm{Out2} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{pmatrix} \begin{pmatrix} \mathrm{In1} \\ \mathrm{In2} \end{pmatrix} \qquad \begin{array}{c|c} \xrightarrow{\mathrm{In1}} & \xleftarrow{\mathrm{In2}} \\ \xleftarrow{\mathrm{Out1}} & \xrightarrow{\mathrm{Out2}} \end{array} \tag{3.37}$$

with the complex coefficients a_{ij}. These matrices serve as a compact and intuitive notation of the coupling coefficients. For solving a set of linear equations, a different notation is more sensible: a linear set of equations can be written in the form of a matrix that represents the interferometer: the *interferometer matrix* times the vector of field amplitudes (solution vector). Together with the right hand side vector that gives numeric values for the input field, the set of linear equations is complete:

$$\begin{pmatrix} \text{interferometer} \\ \text{matrix} \end{pmatrix} \times \begin{pmatrix} \vec{x}_{\mathrm{sol}} \\ \\ \end{pmatrix} = \begin{pmatrix} \vec{x}_{\mathrm{RHS}} \\ \\ \end{pmatrix}. \tag{3.38}$$

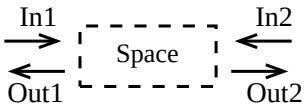
For the above example of a simple mirror the linear set of equations in matrix form looks as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -a_{11} & 1 & -a_{21} & 0 \\ 0 & 0 & 1 & 0 \\ -a_{12} & 0 & -a_{22} & 1 \end{pmatrix} \times \begin{pmatrix} \text{In1} \\ \text{Out1} \\ \text{In2} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} \text{In1} \\ 0 \\ \text{In2} \\ 0 \end{pmatrix}. \tag{3.39}$$

Space

The component 'space' is propagating a light field through free space over a given length L (index of refraction n). The length times index of refraction is by definition (in FINESSE) always a multiple of the default laser wavelength λ_0. This defines a *macroscopic length* (see Section 3.3.2). If the actual length between two other components is not a multiple of the default wavelength, the necessary extra propagation is treated as a feature of one or both end components. For example, a space between two mirrors is always resonant for laser light at the default wavelength. To tune the cavity away from it, one or both mirrors have to be *tuned* (see Section 3.3.2) accordingly while the length of the component 'space' is not changed.

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & s_1 \\ s_2 & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix}$$



The propagation only affects the phase of the field:

$$s_1 = s_2 = \exp(-\mathrm{i}\,\omega n L/c) = \exp(-\mathrm{i}\,(\omega_0 + \Delta\omega) n L/c) = \exp(-\mathrm{i}\,\Delta\omega n L/c), \tag{3.40}$$

where $\exp(-\mathrm{i}\,\omega_0 n L/c) = 1$ following from the definition of macroscopic lengths (see above). The used parameters are the length L, the index of refraction n, the angular frequency of the light field ω, and the offset to the default frequency $\Delta\omega$.

Mirror

From the definition of the component 'space' that always represents a macroscopic length, follows the necessity to perform microscopic propagations inside the mathematical representation of the components *mirror* and *beam splitter*. In this description the component mirror is always hit at normal incidence. Arbitrary angles of incidence are discussed for the component beam splitter, see below.

A light field E_{in} reflected by a mirror is in general changed in phase and amplitude:

$$E_{\text{refl}} = r \exp(\mathrm{i}\,\varphi)\, E_{\text{in}}, \tag{3.41}$$

where r is the amplitude reflectance of the mirror and $\varphi = 2kx$ the phase shift acquired by the propagation towards and back from the mirror if the mirror is not located at the reference plane ($x = 0$).

The *tuning* ϕ gives the displacement of the mirror expressed in radian (with respect to the reference plane). A tuning of $\phi = 2\pi$ represents a displacement of the mirror by one carrier wavelength: $x = \lambda_0$. The direction of the displacement is arbitrarily defined to be in the direction of the normal vector on the front surface, i.e. a positive tuning moves the mirror from node2 towards node1 (for a mirror given by '`m ...node1 node2`').

If the displacement x_m of the mirror is given in meters, then the corresponding tuning ϕ computes as follows:

$$\phi = kx_\mathrm{m} = x_\mathrm{m}\frac{2\pi}{\lambda_0} = x_\mathrm{m}\frac{\omega_0}{c}. \tag{3.42}$$

A certain displacement results in different phase shifts for light fields with different frequencies. The phase shift a general field acquires at the reflection on the front surface of the mirror can be written as:

$$\varphi = 2\phi\frac{\omega}{\omega_0}. \tag{3.43}$$

If a second light beam hits the mirror from the other direction the phase change φ_2 with respect to the same tuning would be:

$$\varphi_2 = -\varphi. \tag{3.44}$$

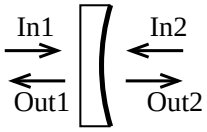
The tuning of a mirror or beam splitter does not represent a change in the path length but a change in the position of component. The transmitted light is thus not affected by the tuning of the mirror (the optical path for the transmitted light always has the same length for all tunings). Only the phase shift of $\pi/2$ for every transmission (as defined in Section 3.3.1) has to be taken into account:

$$E_\mathrm{trans} = \mathrm{i}\ t\ E_\mathrm{in}, \tag{3.45}$$

with t as the amplitude transmittance of the mirror.

The coupling matrix for a mirror is:

$$\begin{pmatrix} \mathrm{Out1} \\ \mathrm{Out2} \end{pmatrix} = \begin{pmatrix} m_{11} & m_{21} \\ m_{12} & m_{22} \end{pmatrix} \begin{pmatrix} \mathrm{In1} \\ \mathrm{In2} \end{pmatrix} \tag{3.46}$$



with the coefficients given as:

$$\begin{aligned} m_{12} &= m_{21} = \mathrm{i}t, \\ m_{11} &= r\ \exp(\mathrm{i}\,2\phi\ \omega/\omega_0), \\ m_{22} &= r\ \exp(-\mathrm{i}\,2\phi\ \omega/\omega_0), \end{aligned}$$

with $\phi = 2\pi$ `phi`$/360$, `phi` as the tuning of the mirror given in the input file, and ω the angular frequency of the reflected light.

Beam splitter

A beam splitter is similar to a mirror except for the extra parameter α which indicates the angle of incidence of the incoming beams and that it can be connected to four nodes. The order in which these nodes have to be entered is shown in Figure 3.4.4.

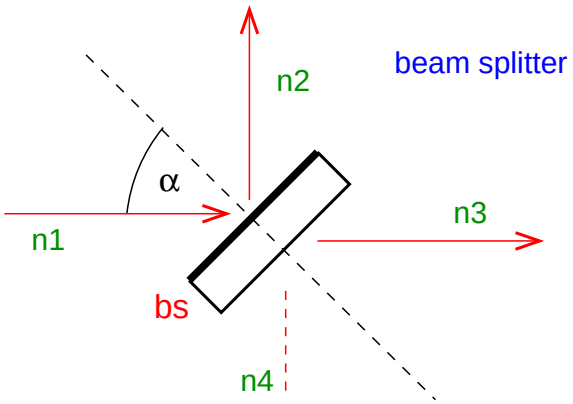


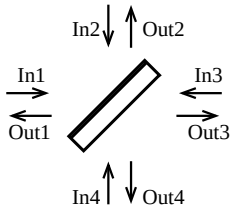
Figure 3.3: *Beam splitter.*

Since, in this work, a displacement of the beam splitter is assumed to be perpendicular to its optical surface, the angle of incidence affects the phase change of the reflected light. Simple geometric calculations lead to the following equation for the optical phase change φ:

$$\varphi \;=\; 2\,\phi\,\frac{\omega}{\omega_0}\cos(\alpha). \tag{3.47}$$

The coupling matrix has the following form:

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \\ \text{Out3} \\ \text{Out4} \end{pmatrix} = \begin{pmatrix} 0 & bs_{21} & bs_{31} & 0 \\ bs_{12} & 0 & 0 & bs_{42} \\ bs_{13} & 0 & 0 & bs_{43} \\ 0 & bs_{24} & bs_{34} & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \\ \text{In3} \\ \text{In4} \end{pmatrix} \tag{3.48}$$



with the coefficients:

$$\begin{aligned} bs_{12} &= bs_{21} = r\ \exp\left(\mathrm{i}\,2\phi\omega/\omega_0\cos\alpha\right), \\ bs_{13} &= bs_{31} = \mathrm{i}\,t, \\ bs_{24} &= bs_{42} = \mathrm{i}\,t, \\ bs_{34} &= bs_{43} = r\ \exp\left(-\mathrm{i}\,2\phi\omega/\omega_0\cos\alpha\right), \end{aligned}$$

and $\phi = 2\pi$ `phi`$/360$.

Modulator

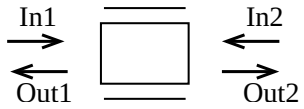
The modulation of light fields is described in Section 3.4.3. A small modulation of a light field in amplitude or phase can be described as follows: a certain amount of light power is shifted from the carrier into new frequency components (sidebands). In general, a modulator can create a very large number of sidebands if, for example, the modulator is located inside a cavity: on every round trip the modulator would create new sidebands around the previously generated sidebands. This effect *cannot* be modelled by the formalism described here.

Instead, a simplified modulator scheme is used. The modulator only acts on specially selected light fields and generates a well-defined number of sidebands. With these simplifications the modulator can be described as:

- an attenuator for the light field that experiences the modulation (at the carrier frequency);
- a source of light at a new frequency (the sideband frequencies), see Section 3.4.5.

All other frequency components of the light field are attenuated by the modulator in accordance with the J_0 Bessel coefficient. The coupling matrix for the modulator is:

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & eo_{21} \\ eo_{12} & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix}$$



The modulation and signal sidebands are not affected by the modulator. The coupling coefficients for field amplitudes at these frequencies are simply:

$$eo_{12} = eo_{21} = 1. \tag{3.49}$$

When the input field is a laser field, the modulator shifts power from the main beam to generate the modulation sidebands. Therefore a modulator reduces the amplitude of the initial field. The phase is not changed:

$$eo_{12} = eo_{21} = C, \tag{3.50}$$

with

$$C = 1 - \frac{m}{2}, \tag{3.51}$$

(m is the modulation index `midx`) for amplitude modulation and

$$C = J_0(m), \tag{3.52}$$

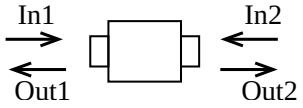
for phase modulation. If the 'single sideband' mode is used, then C is replaced by C':

$$C' = 1 - \frac{1 - C}{2}. \tag{3.53}$$

Isolator (diode)

The isolator represents a simplified Faraday isolator: light passing in one direction is not changed, whereas the power of the beam passing in the other direction is reduced by a specified amount:

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & d_{21} \\ d_{12} & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix} \qquad (3.54)$$



The amplitude coupling coefficients are:

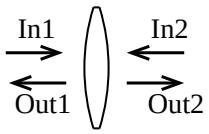
$$\begin{aligned} d_{12} &= 1, \\ d_{21} &= 10^{-S/20}, \end{aligned}$$

with S the specified suppression given in dB.

Lens

The thin lens does not change the amplitude or phase of the light fields.

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix}$$



Gratings

Gratings are optical components which require a more complex treatment than the components above. The context of FINESSE allows simulation of certain aspects of a grating in a well defined configuration. This section gives a short introduction to the implementation of gratings in FINESSE. This work has been done with help by Alexander Bunkowski and the notation is based on his paper [Bunkowski01].

The name grating is used for various very different types of optical components. The following description is restricted to phase gratings used in reflection. However, the implemented formalism can also be used to simulate some properties of optical setups with other grating types.

This phase grating in reflection has been chosen because it can possibly be manufactured with similar optical and mechanical qualities as the high quality mirrors used in gravitational wave detectors today. Thus low-loss laser interferometers with an all-reflective topology can be envisaged.

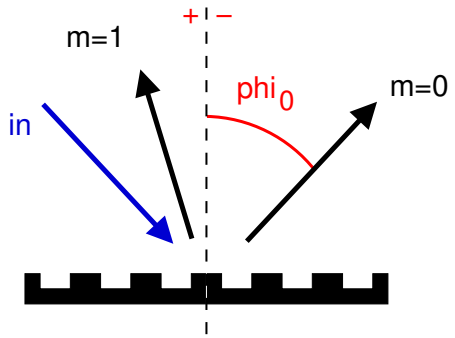


Figure 3.4: *A grating illuminated by a beam (in). The number of outgoing beams is given by the grating equation Equation 3.55. The beams are numbered by an integer (m) and the angles with respect to the grating normal are given as ϕ_m (angles left of the grating normal are positive, right of the normal negative). The geometry is chosen such that the angle of incidence is always positive. $m = 0$ marks the zeroth order reflection which corresponds to a normal reflection. Thus $\phi_0 = -\alpha$. The beams with negative orders ($m = -1, -2, \ldots$) are with angles $-\pi/2 < \phi < -\alpha$ whereas positive orders have angles of $\pi/2 > \phi > -\alpha$. In the example shown here only two orders exist.*

The gratings in FINESSE are characterised by the number of ports. In general a grating is defined by its grating period, given in [nm]. With the wavelength and the angle of incidence α all possible outgoing beams can be computed with the grating equation:

$$\sin(\phi_m) + \sin(\alpha) = \frac{m\,\lambda}{d}, \tag{3.55}$$

with m an integer to label the order of the outgoing beam. An example is shown in Figure 3.4. The geometry is chosen so that always $\alpha \geq 0$. This is possible since the setup is (so far) symmetric. All orders with angles ϕ_m between 90° and −90° exist and will contain some amount of light power. The zeroth order represents the reflection as on a mirror surface with $\phi_0 = -\alpha$. Orders with negative number leave the grating with an angle $-90 < \phi < -\alpha$, positive orders have angles with $90 > \phi > -\alpha$.

In order to construct a device with a small number of ports the grating period has to be chosen such that $d \approx \lambda$. Equation 3.55 can be used to compute limits for the grating parameters with respect to the configuration used. We can write Equation 3.55 as:

$$\frac{m\,\lambda}{d} = \sin(\phi_m) + \sin(\alpha) = [-1, 2]. \tag{3.56}$$

In all following cases more than just the zeroth order ($m = 0$) should exist, n shall be a positive integer. For the positive order $m = n$ to be allowed we get:

$$\frac{\lambda}{d} \leq \frac{2}{n}. \tag{3.57}$$

And the negative order $m = -n$ can exist only if:

$$\frac{\lambda}{d} \leq \frac{1}{n}. \tag{3.58}$$

Table 3.1 gives on overview of the modes that the grating equation allows to exist in certain intervals. Note that we have not yet specified α. Whether an order exists or not can be completely determined only for a given angle of incidence.

m / λ/d	0	1	2	3	4	-1	-2	-3	number of orders
$]2, \text{inf}]$	x								1
$]1, 2]$	x	x							2
$]\frac{2}{3}, 1]$	x	x	x			x			4
$]\frac{1}{2}, \frac{2}{3}]$	x	x	x	x		x			5
$]\frac{1}{3}, \frac{1}{2}]$	x	x	x	x	x	x	x		7
$]\frac{2}{5}, \frac{1}{3}]$	x	x	x	x	x	x	x	x	8

Table 3.1: *Possible existing orders with (for a well-chosen angle of incidence α) in dependence of λ/d.*

Littrow configuration one special setup is the *Littrow* configuration in which the angle of incidence coincides with one mode angle. The nth order Littrow configuration is given by:

$$\phi_n = \alpha, \tag{3.59}$$

which yields:

$$\sin(\alpha) = \frac{n\lambda}{2d}. \tag{3.60}$$

Grating components in Finesse FINESSE offers the following three grating types:

gr2 : a 2 port grating in first order Littrow configuration
gr3 : a 3 port grating in second order Littrow configuration
gr4 : a 4 port device, only the first order exists and is used **not** in Littrow configuration

Each configuration corresponds to a set of limits, for example, the angle of incidence.

In the following, these limits and the coupling matrices for these grating configurations are given. The matrix is given in the form:

$$b_i = A_{ij} a_j, \tag{3.61}$$

with a_j being the vector of incoming fields and b_i the vector of outgoing fields.

gr2 component a grating in first order Littrow configuration is defined by the fact that only the orders $m = 0, 1$ exist and that $\phi_1 = \alpha$. The grating equation therefore reduces to

$$\sin(\phi_m) = (m - 1/2)\lambda/d. \tag{3.62}$$

The existence of $m = 1$ gives $\lambda/d < 2$, the non-existence of $m = -1$ or $m = 2$ yields $\lambda/d > 2/3$. Written together, we get:

$$\lambda/2 < d < 3/2\ \lambda. \tag{3.63}$$

The angle of incidence α and the grating period are related as:

$$\alpha = \arcsin\left(\frac{\lambda}{2d}\right). \tag{3.64}$$

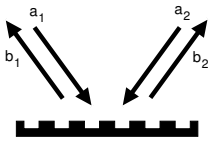
The angle of incidence is set automatically by FINESSE (the positive value is chosen by default).

The two coupling efficiencies η_0, η_1 are constrained by energy conservation[4] (as for the beam splitter) as:

$$\eta_0^2 + \eta_1^2 = 1. \tag{3.65}$$

The coupling matrix is given by:

$$\begin{pmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \end{pmatrix} = \begin{pmatrix} \mathrm{i}\,\eta_1 & \eta_0 \\ \eta_0 & \mathrm{i}\,\eta_1 \end{pmatrix} \begin{pmatrix} \mathrm{a}_1 \\ \mathrm{a}_2 \end{pmatrix}$$



gr3 component The second order Littrow configuration. Only the orders $m = 0, 1, 2$ exist. This gives:

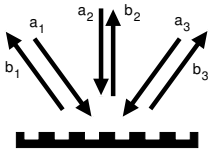
$$\lambda < d < 2\lambda. \tag{3.66}$$

And ϕ_2 must be equal to α. This yields:

$$\alpha = \arcsin\left(\frac{\lambda}{d}\right). \tag{3.67}$$

The coupling matrix for this grating configuration is rather complex [Bunkowski01]. It can be written as:

$$\begin{pmatrix} \mathrm{b}_1 \\ \mathrm{b}_2 \\ \mathrm{b}_3 \end{pmatrix} = \begin{pmatrix} \eta_2 e^{\mathrm{i}\,\phi_2} & \eta_1 e^{\mathrm{i}\,\phi_1} & \eta_0 e^{\mathrm{i}\,\phi_0} \\ \eta_1 e^{\mathrm{i}\,\phi_1} & \rho_0 e^{\mathrm{i}\,\phi_0} & \eta_1 e^{\mathrm{i}\,\phi_1} \\ \eta_0 e^{\mathrm{i}\,\phi_0} & \eta_1 e^{\mathrm{i}\,\phi_1} & \eta_2 e^{\mathrm{i}\,\phi_2} \end{pmatrix} \begin{pmatrix} \mathrm{a}_1 \\ \mathrm{a}_2 \\ \mathrm{a}_3 \end{pmatrix}$$



[4] The gratings are defined as lossless components in FINESSE. This corresponds to the employed phase relations between different orders. Currently, losses can be added only be inserting extra mirrors with $R = 0, T \neq 1$.

with:

$$\begin{aligned} \phi_0 &= 0, \\ \phi_1 &= -\tfrac{1}{2}\arccos\left(\tfrac{\eta_1^2-2\eta_0^2}{2\rho_0\eta_0}\right), \\ \phi_2 &= \arccos\left(\tfrac{-\eta_1^2}{2\eta_2\eta_0}\right). \end{aligned} \tag{3.68}$$

The coupling efficiencies are limited by energy conservation to:

$$\begin{aligned} &\rho_0^2 + 2\eta_1^2 = 1, \\ &\eta_0^2 + \eta_1^2 + \eta_2^2 = 1. \end{aligned} \tag{3.69}$$

Further limits for the coupling efficiencies follow from the coupling phases:

$$\frac{1-\rho_0}{2} \leq \eta_0, \eta_2 \leq \frac{1+\rho_0}{2}. \tag{3.70}$$

gr4 component The grating is *not* used in any Littrow configuration. Only two orders are allowed to exist. From the grating equation one can see that these can only be $m = 0, 1$. To compute the limits for λ/d and α we rewrite the grating equation as:

$$a + b = mc, \tag{3.71}$$

with $a = \sin(\alpha) \in [0, 1]$, $b = \sin(\phi) \in [-1, 1]$ and $c = \lambda/d > 0$. From the fact that the first order $m = 1$ should exist we get:

$$a + b = c. \tag{3.72}$$

This can only be true if

$$c < 2, \tag{3.73}$$

and

$$a > c - 1. \tag{3.74}$$

We can derive the next limit from the fact that $m = -1$ must not exist, i.e.:

$$a + b \neq -c, \tag{3.75}$$

this is true if

$$a + c < -1 \qquad \vee \qquad a + c > 1. \tag{3.76}$$

The first condition is never fulfilled so we get the remaining limit as:

$$a > 1 - c. \tag{3.77}$$

Now, we must make sure that $m = 2$ does not exist:

$$a + b \neq 2c. \tag{3.78}$$

As before we can write this as

$$a < 2c - 1 \qquad \vee \qquad a > 2c + 1. \tag{3.79}$$

The second condition can never be fulfilled. Also the first limit immediately gives $c > \frac{1}{2}$ but together with Equation 3.77 we can restrict possibles values for c even further. Combining Equation 3.77 and Equation 3.79 we get:

$$a > 1 - c \qquad \wedge \qquad a < 2c - 1. \tag{3.80}$$

This is only possible if

$$1 - c < 2c - 1, \tag{3.81}$$

and thus $c > 2/3$.

In summary we get:

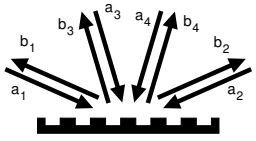
$$\begin{array}{lllll} \lambda/2 & < & d & < & 3/2\ \lambda, \\ 1 - \lambda/d & < & \sin(\alpha) & < & 2\lambda/d - 1 \qquad \text{for} \qquad \lambda/d < 1, \\ \lambda/d - 1 & < & \sin(\alpha) & & \text{for} \qquad \lambda/d > 1. \end{array} \tag{3.82}$$

The coupling of the field amplitude then corresponds to that of a beam splitter. The two coupling efficiencies η_0, η_1 are constrained by energy conservation as:

$$\eta_0^2 + \eta_1^2 = 1. \tag{3.83}$$

The coupling matrix is given by:

$$\begin{pmatrix} \mathrm{b1} \\ \mathrm{b2} \\ \mathrm{b3} \\ \mathrm{b4} \end{pmatrix} = \begin{pmatrix} 0 & A_{21} & A_{31} & 0 \\ A_{12} & 0 & 0 & A_{42} \\ A_{13} & 0 & 0 & A_{43} \\ 0 & A_{24} & A_{34} & 0 \end{pmatrix} \begin{pmatrix} \mathrm{a1} \\ \mathrm{a2} \\ \mathrm{a3} \\ \mathrm{a4} \end{pmatrix}$$



with the coefficients:

$$A_{12} = A_{21} = \eta_0, \tag{3.84}$$
$$A_{13} = A_{31} = \mathrm{i}\,\eta_1, \tag{3.85}$$
$$A_{24} = A_{42} = \mathrm{i}\,\eta_1, \tag{3.86}$$
$$A_{34} = A_{43} = \eta_0. \tag{3.87}$$

3.4.5 Input fields or the 'right hand side' vector

After the set of linear equations for an optical system has been determined, the input light fields have to be given by the user. The respective fields are entered into the 'right hand side' (RHS) vector of the set of linear equations. The RHS vector consists of complex numbers that specify the amplitude and phase of every input field. Input fields are initially set to zero, and every non-zero entry describes a light source. The possible sources are lasers, modulators and 'signal sidebands'.

Laser

The principal light sources are, of course, the lasers. They are connected to one node only. The input power is specified by the user in the input file. For every laser the field amplitude is set as:

$$a_{in} = \sqrt{(P/\epsilon_c)}\; e^{\mathrm{i}\,\varphi}, \tag{3.88}$$

with

$$P \;=\; \epsilon_0 c |a|^2, \tag{3.89}$$

as the laser power and φ the specified phase. The conversion factor $\epsilon_c = \epsilon_0 \cdot c$ can be set in the init file 'kat.ini'. The default value is $\epsilon_c = 1$. This setting does not yield correct absolute values for light field amplitudes, i.e. when amplitude detectors are used. Instead, one obtains more intuitive numbers from which the respective light power can be easily computed. For the correct units of field amplitudes, the value for ϵ_c can be set to $\epsilon_c = \epsilon_0 \cdot c \approx 0.0026544$.

Modulators

Modulators produce non-zero entries in the RHS vector for every modulation sideband generated. Depending on the order ($k \geq 0$) and the modulation index (m), the input field amplitude for amplitude modulation is:

$$a_{\mathrm{in}} = \frac{m}{4}, \tag{3.90}$$

and for phase modulation:

$$a_{\mathrm{in}} = (-1)^k \; J_k(m) \;\; \exp(\mathrm{i}\,\varphi), \tag{3.91}$$

with φ given as (Equation 3.12):

$$\varphi = \pm k \cdot (\frac{\pi}{2} + \varphi_{\mathrm{s}}), \tag{3.92}$$

where φ_{s} is the user-specified phase from the modulator description. The sign of φ is the same as the sign of the frequency offset of the sideband. For 'lower' sidebands ($f_{\mathrm{mod}} < 0$) we get $\varphi = -\ldots$, for 'upper' sidebands ($f_{\mathrm{mod}} > 0$) it is $\varphi = +\ldots$.

Signal frequencies

The most complex input light fields are the signal sidebands. They can be generated by many different types of modulation inside the interferometer (signal modulation in the following). The components mirror, beam splitter, space, laser and modulator can be used as a source of signal sidebands. Primarily, signal sidebands are used as the input signal for computing transfer functions of the optical system. The amplitude, in fact the modulation index, of the signal is assumed to be much smaller than unity so that the effects of the modulation can be described by a linear analysis. If linearity is assumed, however, the computed transfer functions are independent of the signal amplitude; thus, only the relative amplitudes of output and input are important, and the modulation index of the signal modulation can be arbitrarily set to unity in the simulation.

Signal frequencies can be 'applied' to a number of different components using the command `fsig`. The connection of the signal frequency causes the component to—in some way—modulate the light fields at the component. The frequency, amplitude and phase of the modulation can be specified by `fsig`.

FINESSE always assumes a numerical signal amplitude of 1. The numerical value of 1 has a different meaning for applying signals to different components (see below). The amplitude specified with `fsig` can be used to define the relative amplitudes of the source when the signal is applied to several components at once. Please note that FINESSE does not correct the transfer functions for strange amplitude settings. An amplitude setting of two, for example, will scale the output (a transfer function) by a factor of two.

In order to have a determined number of light fields, the signal modulation of a signal sideband has to be neglected. This approximation is sensible because in the steady state the signal modulations are expected to be tiny so that second-order effects (signal modulation of the signal modulation fields) can be omitted.

In general, the carrier field at the 'signal component' can be written as:

$$E_{in} \;=\; E_0 \;\exp\left(\mathrm{i}\,\omega_c\,t + \varphi_c\right), \tag{3.93}$$

with ω_c the carrier frequency, and φ_c the phase of the carrier. In most cases the modulation of the light will be a phase modulation. Then the field after the modulation can be expressed in general as:

$$E_{out} \;=\; A\;E_0 \;\exp\left(\mathrm{i}\,\omega_c\,t + \varphi_c + \varphi(t) + B\right), \tag{3.94}$$

with A as a real amplitude factor, B a constant phase term and

$$\varphi(t) \;=\; m\cos\left(\omega_s\,t + \varphi_s\right), \tag{3.95}$$

with m the modulation index, ω_s the signal frequency and φ_s the phase as defined by `fsig`. The modulation index will in general depend on the signal amplitude a_s as given by `fsig` and also other parameters (see below). As mentioned in Section 3.1.2, the simple

form for very small modulation indices ($m \ll 1$) can be used: only the two sidebands of the first order are taken into account and the Bessel functions can be approximated by:

$$\begin{aligned} J_0(m) &\approx 1, \\ J_{\pm 1}(m) &\approx \pm \tfrac{m}{2}. \end{aligned} \tag{3.96}$$

Thus, the modulation results in two sidebands ('upper' and 'lower' with a frequency offset of $\pm\omega_s$ to the carrier) which can be written as:

$$\begin{aligned} E_{sb} &= \mathrm{i}\,\tfrac{m}{2}\; A\; E_0\; \exp\left(\mathrm{i}\left((\omega_c \pm \omega_s)\,t + \varphi_c + B \pm \varphi_s\right)\right) \\ &= \tfrac{m}{2}\; A\; E_0\; \exp\left(\mathrm{i}\left(\omega_{sb}\,t + \pi/2 + \varphi_c + B \pm \varphi_s\right)\right). \end{aligned} \tag{3.97}$$

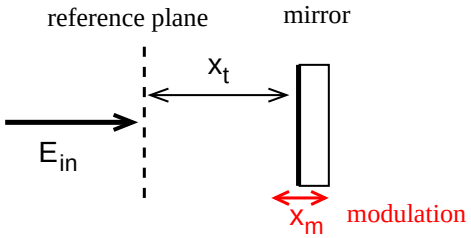


Figure 3.5: *Signal applied to a mirror: modulation of the mirror position.*

Mirror Mirror motion does not change transmitted light. The reflected light will be modulated in phase by any mirror movement. The relevant parameters are shown in Figure 3.5. At a reference plane (at the nominal mirror position when the tuning is zero), the field impinging on the mirror is:

$$E_{\mathrm{in}} \;=\; E_0\; \exp\Big(\mathrm{i}\left(\omega_{\mathrm{c}}t + \varphi_{\mathrm{c}} - k_{\mathrm{c}}x\right)\Big) \;=\; E_0\; \exp\left(\mathrm{i}\,\omega_{\mathrm{c}}t + \varphi_{\mathrm{c}}\right). \tag{3.98}$$

If the mirror is detuned by x_{t} (here given in meters) then the electric field at the mirror is:

$$E_{\mathrm{mir}} \;=\; E_{\mathrm{in}}\; \exp\left(-\mathrm{i}\,k_{\mathrm{c}}x_{\mathrm{t}}\right). \tag{3.99}$$

With the given parameters for the signal frequency, the position modulation can be written as $x_{\mathrm{m}} = a_{\mathrm{s}}\cos(\omega_{\mathrm{s}}t + \varphi_{\mathrm{s}})$ and thus the reflected field at the mirror is:

$$E_{\mathrm{refl}} \;=\; r\; E_{\mathrm{mir}}\; \exp\left(\mathrm{i}\,2k_{\mathrm{c}}x_{\mathrm{m}}\right) \;=\; r\; E_{\mathrm{mir}}\; \exp\Big(\mathrm{i}\,2k_{\mathrm{c}}a_{\mathrm{s}}\cos(\omega_{\mathrm{s}}t + \varphi_{\mathrm{s}})\Big), \tag{3.100}$$

setting $m = 2k_{\mathrm{c}}a_{\mathrm{s}}$, this can be expressed as:

$$\begin{aligned} E_{\mathrm{refl}} &= r\; E_{\mathrm{mir}}\; \Big(1 + \mathrm{i}\,\tfrac{m}{2}\exp\Big(-\mathrm{i}\left(\omega_{\mathrm{s}}t + \varphi_{\mathrm{s}}\right)\Big) + \mathrm{i}\,\tfrac{m}{2}\exp\Big(\mathrm{i}\left(\omega_{\mathrm{s}}t + \varphi_{\mathrm{s}}\right)\Big)\Big) \\ &= r\; E_{\mathrm{mir}}\; \Big(1 + \tfrac{m}{2}\exp\Big(-\mathrm{i}\left(\omega_{\mathrm{s}}t + \varphi_{\mathrm{s}} - \pi/2\right)\Big) \\ &\quad + \tfrac{m}{2}\exp\Big(\mathrm{i}\left(\omega_{\mathrm{s}}t + \varphi_{\mathrm{s}} + \pi/2\right)\Big)\Big). \end{aligned} \tag{3.101}$$

This gives an amplitude for both sidebands of:

$$a_{\mathrm{sb}} \;= r\; m/2\; E_0 \;=\; r\; k_{\mathrm{c}} a_{\mathrm{s}}\; E_0. \tag{3.102}$$

The phase back at the reference plane is:

$$\varphi_{\mathrm{sb}} \;=\; \varphi_{\mathrm{c}} + \frac{\pi}{2} \pm \varphi_{\mathrm{s}} - (k_{\mathrm{c}} + k_{\mathrm{sb}})\; x_{\mathrm{t}}, \tag{3.103}$$

where the plus sign refers to the 'upper' sideband and the minus sign to the 'lower' sideband. As in FINESSE the tuning is given in degrees, i.e. the conversion from x_{t} to ϕ has to be taken into account:

$$\begin{aligned} \varphi_{\mathrm{sb}} &= \varphi_{\mathrm{c}} + \tfrac{\pi}{2} \pm \varphi_{\mathrm{s}} - (\omega_{\mathrm{c}} + \omega_{\mathrm{sb}})/c\; x_{\mathrm{t}} \\ &= \varphi_{\mathrm{c}} + \tfrac{\pi}{2} \pm \varphi_{\mathrm{s}} - (\omega_{\mathrm{c}} + \omega_{\mathrm{sb}})/c\; \lambda_0/360\; \phi \\ &= \varphi_{\mathrm{c}} + \tfrac{\pi}{2} \pm \varphi_{\mathrm{s}} - (\omega_{\mathrm{c}} + \omega_{\mathrm{sb}})/\omega_0\; 2\pi/360\; \phi. \end{aligned} \tag{3.104}$$

With a nominal signal amplitude of $a_{\mathrm{s}} = 1$, the sideband amplitudes become very large. For an input light field at the default wavelength one typically obtains:

$$a_{\mathrm{sb}} = r\; k_{\mathrm{c}}\; E_0 = r\; \omega_{\mathrm{c}}/c\; E_0 = r\; 2\pi/\lambda_0\; E_0 \approx 6 \cdot 10^6. \tag{3.105}$$

Numerical algorithms have the best accuracy when the various input numbers are of the same order of magnitude, usually set to a number close to one. Therefore, the signal amplitudes for mirrors (and beam splitters) should be scaled: a natural scale is to define the modulation in radians instead of meters. The scaling factor is then ω_0/c, and setting $a = \omega_0/c\; a'$ the reflected field at the mirror becomes:

$$\begin{aligned} E_{\mathrm{refl}} &= r\; E_{\mathrm{mir}}\; \exp\left(\mathrm{i}\, 2\omega_{\mathrm{c}}/\omega_0\; x_{\mathrm{m}}\right) \\ &= r\; E_{\mathrm{mir}}\; \exp\Big(\mathrm{i}\, 2\omega_{\mathrm{c}}/\omega_0\; a'_{\mathrm{s}} \cos(\omega_{\mathrm{s}} t + \varphi_{\mathrm{s}})\Big), \end{aligned} \tag{3.106}$$

and thus the sideband amplitudes are:

$$a_{\mathrm{sb}} \;= r\;\; \omega_{\mathrm{c}}/\omega_0\; a'_{\mathrm{s}}\; E_0, \tag{3.107}$$

with the factor $\omega_{\mathrm{c}}/\omega_0$ typically being close to one. The units of the computed transfer functions are 'output unit per radian'; which are neither common nor intuitive. The command `scale meter` converts the units into the more common 'Watts per meter' by applying the inverse scaling factor c/ω_0.

When a light field is reflected at the back surface of the mirror, the sideband amplitudes are computed accordingly. The same formulae as above can be applied with $x_{\mathrm{m}} \to -x_{\mathrm{m}}$ and $x_{\mathrm{t}} \to -x_{\mathrm{t}}$, yielding the same amplitude as for the reflection at the front surface, but with a slightly different phase:

$$\begin{aligned} \varphi_{\mathrm{sb,back}} &= \varphi_{\mathrm{c}} + \tfrac{\pi}{2} \pm (\varphi_{\mathrm{s}} + \pi) + (k_{\mathrm{c}} + k_{\mathrm{sb}})\; x_{\mathrm{t}} \\ &= \varphi_{\mathrm{c}} + \tfrac{\pi}{2} \pm (\varphi_{\mathrm{s}} + \pi) + (\omega_{\mathrm{c}} + \omega_{\mathrm{sb}})/\omega_0\; 2\pi/360\; \phi. \end{aligned} \tag{3.108}$$

Beam splitter When the signal frequency is applied to the beam splitter, the reflected light is modulated in phase. In fact, the same computations as for mirrors can be used for beam splitters. However, all distances have to be scaled by $\cos(\alpha)$ (see Section 3.4.4). Again, only the reflected fields are changed by the modulation and the front side and back side modulation have different phases. The amplitude and phases compute to:

$$\begin{aligned} a_{\mathrm{sb}} &= r \;\; \omega_c/\omega_0 \; a_{\mathrm{s}} \cos(\alpha) \; E_0, && (3.109)\\ \phi_{\mathrm{sb,front}} &= \varphi_c + \frac{\pi}{2} \pm \varphi_{\mathrm{s}} - (k_c + k_{sb}) x_t \cos(\alpha), && (3.110)\\ \phi_{\mathrm{sb,back}} &= \varphi_c + \frac{\pi}{2} \pm (\varphi_{\mathrm{s}} + \pi) + (k_c + k_{sb}) x_t \cos(\alpha). && (3.111) \end{aligned}$$

Space For interferometric gravitational wave detectors, the 'free space' is an important source for a signal frequency: a passing gravitational wave modulates the length of the space (i.e. the distance between two test masses). A light field passing this length will thus be modulated in phase. The phase change $\phi(t)$ which is accumulated over the full length is (see, for example, [Mizuno]):

$$\phi(t) = \frac{\omega_c \; n \; L}{c} + \frac{a_s}{2} \frac{\omega_c}{\omega_s} \sin\left(\omega_s \frac{n \; L}{c}\right) \cos\left(\omega_s \left(t - \frac{n \; L}{c}\right)\right), \tag{3.112}$$

with L the length of the space, n the index of refraction and a_s the signal amplitude given in strain (h). This results in a signal sideband amplitude of:

$$\begin{aligned} a_{\mathrm{sb}} &= \frac{1}{4} \frac{\omega_c}{\omega_s} \sin\left(\omega_s \frac{n \; L}{c}\right) a_{\mathrm{s}} \; E_0, && (3.113)\\ \phi_{\mathrm{sb}} &= \varphi_c + \frac{\pi}{2} \pm \varphi_s - (\omega_c + \omega_s) \frac{nL}{c}. && (3.114) \end{aligned}$$

Laser Applying a signal to a laser is treated as a frequency modulation of the laser:

$$E = E_0 \; e^{\mathrm{i}\,(\omega_c\, t + a_s/\omega_s \cos(\omega_s\, t + \varphi_s) + \varphi_c)}. \tag{3.115}$$

Therefore the amplitude of the sidebands is scaled with frequency as:

$$\begin{aligned} a_{sb} &= \tfrac{a_{\mathrm{s}}}{2\omega_{\mathrm{s}}} \; E_0, \\ \phi_{\mathrm{sb}} &= \varphi_c + \tfrac{\pi}{2} \pm \varphi_s. \end{aligned} \tag{3.116}$$

Modulator Signal frequencies at a modulator are treated as 'phase noise'. See Section 3.4.3 for details. The electric field that leaves the modulator can be written as:

$$\begin{aligned} E = E_0 \;\; & e^{\mathrm{i}\,(\omega_0\, t + \varphi_0)} \; \textstyle\sum_{k=-order}^{order} i^{\,k} \; J_k(m) \; e^{\mathrm{i}\,k(\omega_m\, t + \varphi_m)} \\ & \times \left(1 + \mathrm{i}\, \tfrac{k \; m_2 \; a_s}{2} \; e^{-i(\omega_s\, t + \varphi_s)} \; + \mathrm{i}\, \tfrac{k \; m_2 \; a_s}{2} \; e^{\mathrm{i}\,(\omega_s\, t + \varphi_s)} + O((km_2)^2)\right), \end{aligned} \tag{3.117}$$

with

$$\begin{aligned} E_{\mathrm{mod}} &= E_0\ J_k(m), \\ \varphi_{\mathrm{mod}} &= \varphi_0 + k\tfrac{\pi}{2} + k\varphi_m. \end{aligned} \tag{3.118}$$

The sideband amplitudes are:

$$\begin{aligned} a_{sb} &= \tfrac{a_s k m_2}{2}\ E_{\mathrm{mod}}, \\ \phi_{\mathrm{sb}} &= \varphi_{\mathrm{mod}} + \tfrac{\pi}{2} \pm \varphi_s. \end{aligned} \tag{3.119}$$

3.4.6 Photodetectors and demodulation

With all the different ways of plotting output signals in FINESSE it is important to understand that every photodiode output represents only *one* of many possible component of a signal. The 'output signal' can be a light field, a light power, or a mixer output. Of course, you can calculate more than one component at a time, but only by specifying different detectors for each of them.

When the program has calculated the light field amplitudes for every frequency and every output port, the interferometer is completely 'solved'. With the field amplitudes and phases you can now calculate every error signal or frequency response. The different possible detectors you can use with FINESSE are meant to simplify this task for the most common purposes. Each photodetector type calculates a special output from the available field amplitudes. The following paragraphs show how this is done. For simplicity the calculations will be given for one output port only.

Common to several detector types are some scaling factors which can be applied: the command `scale` can be used to scale the output by a given factor. Several preset factors can be used:

- `scale ampere output` in the input file scales light power to photocurrent for the specified detector. The scaling factor (from Watts to Amperes) is:

$$C_{\mathrm{ampere}} = \frac{e\ q_{\mathrm{eff}}\ \lambda_0}{hc}\ [\frac{\mathrm{A}}{\mathrm{W}}], \tag{3.120}$$

  with e the electron charge, q_{eff} the quantum efficiency of the detector, h Planck's constant, and c the speed of light. λ_0 is the default laser wavelength; if several light fields with different wavelengths are present or sidebands are concerned, still only one wavelength is used in this calculation. The differences in λ should be very small in most cases so that the resulting error is negligible.
- `scale meter output` can be useful when a transfer function has been computed and `fsig` was applied to a mirror or beam splitter. The output is scaled by $2\pi/\lambda_0$ (or $\lambda_0/2\pi$ for `pdS`). Because the microscopic movement of these components is always set via the tuning, a computed transfer function with the signal inserted at a mirror or beam splitter will have the units Watt/radian. With `scale meter` the result will be rescaled to Watt/meter. In case of a sensitivity (`pdS`), the output will be scaled to $\mathrm{m}/\sqrt{\mathrm{Hz}}$.

- `scale deg output` will scale the output by $180/\pi$. This may be useful in some cases. For example, if the DC value of a transfer function is to be compared to the slope of an error signal (at the operating point). The latter is usually given in Watt/degree whereas the transfer function is typically Watt/radian.

In general, several light fields with different amplitudes, phases and frequencies will be present on a detector. The resulting light field in an interferometer output (i.e. on a detector) can be written as

$$E = e^{\mathrm{i}\omega_0 t} \sum_{n=0}^{N} a_n \, e^{\mathrm{i}\omega_n t}, \tag{3.121}$$

where the a_n are complex amplitudes.

The frequency ω_0 is the default laser frequency, and ω_n are offset frequencies to ω_0 (either positive, negative or zero). Note that very often a slightly different representation is chosen

$$E = e^{\mathrm{i}\omega_0 t} \left(b_0 + b_1 \, e^{\mathrm{i}\omega_1 t} + b_{-1} \, e^{-\mathrm{i}\omega_1 t} + \cdots + b_M \, e^{\mathrm{i}\omega_M t} + b_{-M} \, e^{-\mathrm{i}\omega_M t} \right), \tag{3.122}$$

where ω_0 is the carrier frequency and $\omega_1, \omega_2, \ldots, \omega_M > 0$ are the (symmetric) sidebands. However, in a general approach there might be more than one carrier field and the sidebands are not necessarily symmetric, so Equation 3.121 is used here.

Amplitude detector

The amplitude detector simply plots the already calculated amplitudes of the light field at the specified frequency. The amplitude at frequency ω_m is a complex number (z), and is calculated as follows:

$$z = \sum_n a_n \quad \text{with} \quad \{n \mid n \in \{0, \ldots, N\} \ \wedge \ \omega_n = \omega_m\}. \tag{3.123}$$

Note that the amplitude detector distinguishes between positive and negative frequencies.

Photodetectors

For real detectors we have to look at the intensity at the output port:

$$\begin{aligned} |E|^2 &= E \cdot E^* = \sum_{i=0}^{N} \sum_{j=0}^{N} a_i a_j^* \, e^{\mathrm{i}\,(\omega_i - \omega_j)\, t} \\ &= A_0 + A_1 e^{\mathrm{i}\,\bar{\omega}_1 t} + A_2 e^{\mathrm{i}\,\bar{\omega}_2 t} + \ldots, \end{aligned} \tag{3.124}$$

with the A_i being the amplitudes of the light power sorted by the beat frequencies $\bar{\omega}_i$. In fact, FINESSE never calculates the light power as above, instead it only calculates parts of it depending on the photodetector you specify. There are basically two ways of using photodetectors in FINESSE: a simple photodetector for DC power and a detector with up to 5 demodulations. These detectors see different parts of the sum in Equation 3.124.

The DC detector looks for all components without frequency dependence. The frequency dependence vanishes when the frequency becomes zero, i.e. in all addends of Equation 3.124 with $\omega_i = \omega_j$. The output is a real number, calculated like this:

$$x = \sum_i \sum_j a_i a_j^* \quad \text{with} \quad \{i,j \mid i,j \in \{0,\dots,N\} \ \wedge \ \omega_i = \omega_j\}. \tag{3.125}$$

A single demodulation can be described by a multiplication of the output with a cosine: $\cos(\omega_x + \varphi_x)$ (ω_x is the demodulation frequency and φ_x the demodulation phase) which is also called the 'local oscillator'. In FINESSE, the term 'demodulation' also implies a low pass filtering of the signal after multiplying it with a local oscillator. After whatever demodulation was performed only the **DC** part of the result is taken into account. The signal is

$$S_0 \ = \ |E|^2 \ = \ E \cdot E^* \ = \ \sum_{i=0}^{N} \sum_{j=0}^{N} a_i a_j^* \ e^{\mathrm{i}\,(\omega_i - \omega_j)\,t}, \tag{3.126}$$

multiplied with the local oscillator it becomes

$$\begin{aligned} S_1 \ &= \ S_0 \cdot \cos(\omega_x t + \varphi_x) \ = \ S_0 \tfrac{1}{2} \left(e^{\mathrm{i}\,(\omega_x t + \varphi_x)} + e^{-\mathrm{i}\,(\omega_x t + \varphi_x)} \right) \\ &= \ \tfrac{1}{2} \sum_{i=0}^{N} \sum_{j=0}^{N} a_i a_j^* \ e^{\mathrm{i}\,(\omega_i - \omega_j)\,t} \cdot \left(e^{\mathrm{i}\,(\omega_x t + \varphi_x)} + e^{-\mathrm{i}\,(\omega_x t + \varphi_x)} \right). \end{aligned} \tag{3.127}$$

With $A_{ij} = a_i a_j^*$ and $e^{\mathrm{i}\,\omega_{ij}\,t} = e^{\mathrm{i}\,(\omega_i - \omega_j)\,t}$ we can write

$$S_1 \ = \ \frac{1}{2} \left(\sum_{i=0}^{N} A_{ii} + \sum_{i=0}^{N} \sum_{j=i+1}^{N} \left(A_{ij} \ e^{\mathrm{i}\,\omega_{ij}\,t} + A_{ij}^* \ e^{-\mathrm{i}\,\omega_{ij}\,t} \right) \right) \cdot \left(e^{\mathrm{i}\,(\omega_x t + \varphi_x)} + e^{-\mathrm{i}\,(\omega_x t + \varphi_x)} \right). \tag{3.128}$$

When looking for the DC components of S_1 we get the following

$$\begin{aligned} S_{1,\mathrm{DC}} \ &= \ \sum_{ij} \tfrac{1}{2} (A_{ij} \ e^{-\mathrm{i}\,\varphi_x} + A_{ij}^* \ e^{\mathrm{i}\,\varphi_x}) \quad \text{with} \quad \{i,j \mid i,j \in \{0,\dots,N\} \ \wedge \ \omega_{ij} = \omega_x\} \\ &= \ \sum_{ij} \mathrm{Re} \left\{ A_{ij} \ e^{-\mathrm{i}\,\varphi_x} \right\}. \end{aligned} \tag{3.129}$$

This would be the output of a mixer. The results for $\varphi_x = 0$ and $\varphi_x = \pi/2$ are called *in-phase* and *in-quadrature* respectively (or also *first* and *second quadrature*). They are given by:

$$\begin{aligned} S_{1,\mathrm{DC,phase}} \ &= \ \sum_{ij} \mathrm{Re} \left\{ A_{ij} \right\}, \\ S_{1,\mathrm{DC,quad}} \ &= \ \sum_{ij} \mathrm{Im} \left\{ A_{ij} \right\}. \end{aligned} \tag{3.130}$$

When the user has specified a demodulation phase the output given by FINESSE is real

$$x = S_{1,\mathrm{DC}}. \tag{3.131}$$

If no phase is given the output is a complex number

$$z = \sum_{ij} A_{ij} \quad \text{with} \quad \{i,j \mid i,j \in \{0,\ldots,N\} \ \wedge \ \omega_{ij} = \omega_x\}. \tag{3.132}$$

A double demodulation is a multiplication with two local oscillators and taking the DC component of the result. First looking at the whole signal we can write:

$$S_2 = S_0 \cdot \cos(\omega_x + \varphi_x)\cos(\omega_y + \varphi_y). \tag{3.133}$$

This can be written as

$$\begin{aligned} S_2 &= S_0 \tfrac{1}{2}(\cos(\omega_y + \omega_x + \varphi_y + \varphi_x) + \cos(\omega_y - \omega_x + \varphi_y - \varphi_x)) \\ &= S_0 \tfrac{1}{2}(\cos(\omega_+ + \varphi_+) + \cos(\omega_- + \varphi_-)), \end{aligned} \tag{3.134}$$

and thus reduced to two single demodulations. Since we now only care for the DC component we can use the expression from above (Equation 3.129). These two demodulations give two complex numbers:

$$\begin{aligned} z1 &= \sum_{ij} A_{ij} \quad \text{with} \quad \{i,j \mid i,j \in \{0,\ldots,N\} \ \wedge \ \omega_i - \omega_j = \omega_+\}, \\ z2 &= \sum_{ij} A_{kl} \quad \text{with} \quad \{k,l \mid k,l \in \{0,\ldots,N\} \ \wedge \ \omega_k - \omega_l = \omega_-\}. \end{aligned} \tag{3.135}$$

The demodulation phases are applied as follows to get a real output (two sequential mixers):

$$x = \mathrm{Re}\left\{(z_1 \ e^{-\mathrm{i}\,\varphi_x} + z_2 \ e^{\mathrm{i}\,\varphi_x}) \ e^{-\mathrm{i}\,\varphi_y}\right\}. \tag{3.136}$$

A demodulation phase for the first frequency (here φ_x) must be given in any case. To get a complex output the second phase can be omitted:

$$z = z_1 \ e^{-\mathrm{i}\,\varphi_x} + z_2 \ e^{\mathrm{i}\,\varphi_x}. \tag{3.137}$$

More demodulations can also be reduced to single demodulations as above. In fact the same code computes output signals for up to 5 demodulations.

3.5 The `lock` command

The `lock` command as described in the Syntax reference can be used to implement a feedback system using an internal iterative process. For each data point as specified by the `xaxis` commands the `lock` command solves the interferometer matrix several times in

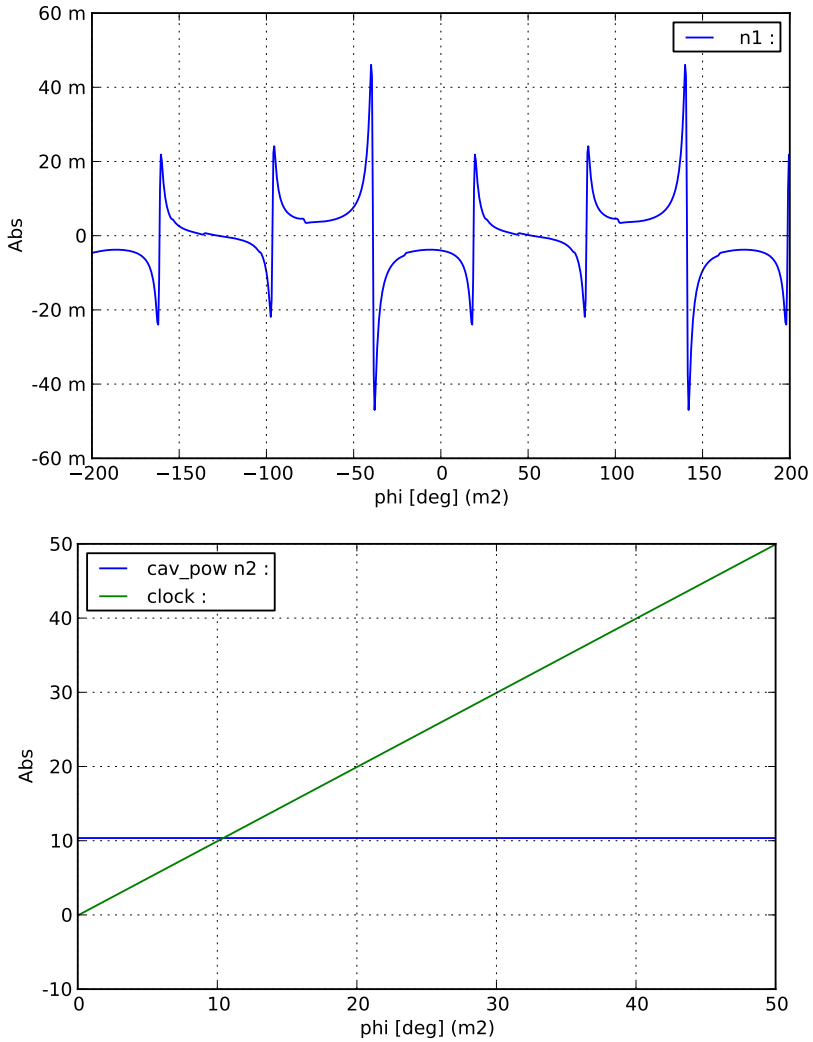


Figure 3.6: *Using a Pound-Drever signal for the* `lock` *command: the top plot shows the error signal, the lower plot shows the correction signal when the lock is active and the cavity circulating power as a test signal. The correction signal 'clock' increases proportional to the detuning as expected and the cavity power is maintained throughout.*

order to perform the iteration. Used correctly the `lock` command can be an effective tool. However, it should be noted that even a simple `lock` often increases the computation time by factors of five.

One of the main uses of the `lock` command is to keep a cavity on resonance in the presence of beam shape distortions, either through mirror misalignments, mode mismatch, or more complex surface distortions introduced via mirror maps, see Section 4.7. In all these cases higher-order modes play a significant role in the description of the optical fields, and as a result the simple trick of making all spaces to be multiples of the wavelength does not

guarantee resonance condition for mirror tunings of $\phi = 0$. Instead the correct mirror tunings need to be set explicitly, for example, by first scanning the cavity and finding the resonance condition. Alternatively a `lock` command can be used to iteratively adjust the tuning of a cavity for each data point.

3.5.1 Using a real error signal for a lock

One of the best ways to ensure that a lock is performing as expected is to use a real error signal, for example to lock a cavity with a Pound-Drever Hall (PDH) error signal, as shown in Figure 3.6. In the presence of higher-order modes, the cavity resonance does not necessarily coincide with the maximum circulating power. If a PDH signal is used in the experiment, you should use the same error signal in FINESSE to ensure that the cavity in the model is always tuned to the same operating point as the experiment. See Appendix A for another short tutorial on how to setup and lock a simple cavity.

This example is on purpose using a mode-mismatched and misaligned cavity to show the impact of these distortions on the error signal. The top plot in Figure 3.6 clearly shows the zero crossings of the carrier and sideband fields. In order to guarantee a successful lock the simulation should be started on resonance, for example we can tune m1 so that the cavity is on resonance and if we chose to tune m2, as shown in the bottom plot, this should start at a tuning of zero. The FINESSE file for using a Pound-Drever Hall signal with the `lock` command is:

```
l i1 1 0 n0
gauss g1 i1 n0 22m -1.2
startnode n0

mod eo1 40k 0.3 3 pm n0 n1      # phase modulator f_mod=40kHz
                                # midx=0.3 order=3
m m1 0.95 0.01 $m1tune n1 n2    # cavity
s s2 1200 n2 n3
m m2 0.99 0.01 0 n3 n4
cav cav1 m1 n2 m2 n3

attr m2 Rc 3000                 # distortions, unmatched
attr m1 xbeta 1u                # curvature and misalignment
phase 0                         # turn off phase adjustement

pd1 inphase 40k 0 n1            # photo diode + mixer
                                # f_demod=40kHz phase=0

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PDH lock of cavity
noplot inphase
set err inphase re
lock clock $err -10 .1m
```

```
%noplot clock
put* m1 phi $clock
pd cav_pow n2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

const m1tune 39
xaxis m2 phi lin 0 50 200
maxtem 3

yaxis abs
```

3.5.2 Setting the `lock` gain

In order to use a lock successfully we must understand the parameters of the optical system. Basically we need to know the operating point, the gain of the error signal at the operating point, the linear range of the error signal and how close we want the interferometer to be at the operating point to call it 'in lock'.

Experience has shown that a good startting point for setting the gain is to aim for a total 'loop gain' of 1. The following example has been taken from a document describing GEO 600 simulations, in particular this is the setup of a longitudinal lock of the Power Recycling degree of freedom.

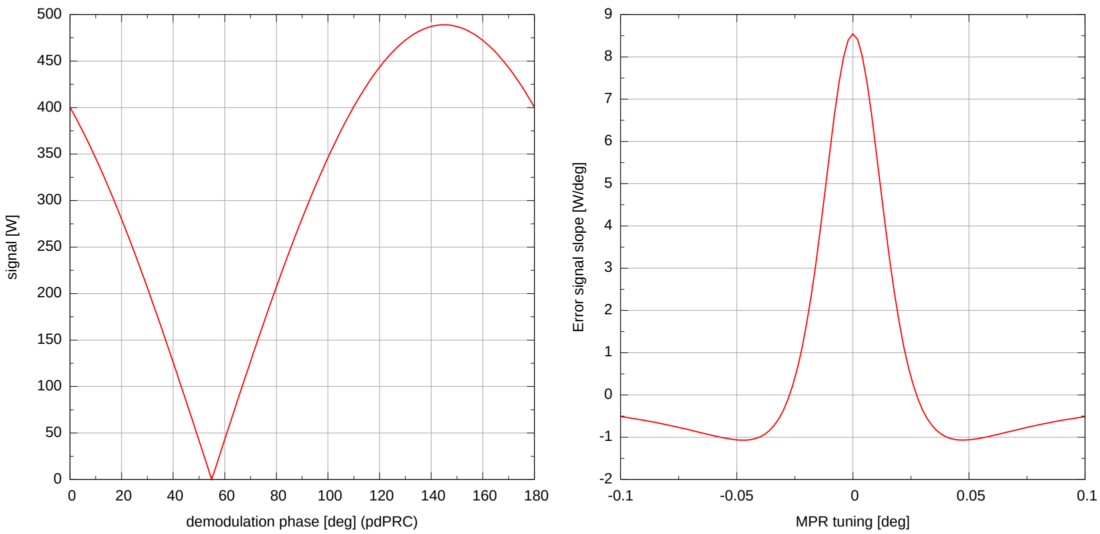


Figure 3.7: left: *PRC error signal as a function of the demodulation phase of pdPRC, maximum signal is obtained at 144.9 deg* **right:** *Slope of the PRC error signal as a function of MPR movement, slope at operating point is 8.5 W/deg*

In this case the operating point at the default Power Recycling mirror (MPR) tuning of $\varphi = 0$ deg. However, it is often a good idea to check the operating point, in this case simply by plotting the cavity power as a function of the MPR tuning. Figure 3.7 shows

the next two steps: first compute the optimal phase for the photodiode. The FINESSE code would be:

```
fsig sig1 MPR 1 0
pd2 pdPRC $fPR 0 1 nBDIPR1
xaxis pdPRC phi1 lin 0 180 200
yaxis abs
```

The slope of the error signal at the operating point is computed with:

```
pd1 pdPRC $fPR 144.9  nBDIPR1
diff MPR phi
xaxis MPR phi lin -0.1 .1 100
```

The slope is 8.5 W/deg. The `lock` commands works best with minus the inverse gain; we use -0.1:

```
pd1 pdPRC $fPR 144.9 nBDIPR1
set errpr pdPRC re
lock prlock $errpr -.1 1m
put* MPR phi $prlock
```

The accuracy given above as `1m` means that the error signal must be smaller than 1 mW. Which correspond (via the gain) to a 10^{-4} tuning of MPR or a displacement of $10^{-4}/360 * \lambda = 0.3\,\mathrm{pm}$.

3.5.3 Tuning the `lock`

You can improve the speed of the `lock` by carefully tuning the following parameters:

- gain: the iteration routines have an 'autogain' feature which tries to correct for wrongly set gain values. However, it can only check for large deviations from optimal gains. For example, setting the gain wrong by a factor of two typically reduces the speed of the simulation by a factor of two or worse.
- locking accuracy: the lock accuracy defines how large the residual deviation between the set locking point and the iterative results should be. Make sure that you know what accuracy you require and do try not to set more stringent values than those required.
- `xaxis` step size: One must make sure that the starting point of the xaxis command is actually at or close to the set point of any active `lock`. Otherwise the `lock` iteration might fail to find the set point.
  Furthermore, in many cases the lock uses on an error signal that does not change linearly with the parameter with tuned by the `xaxis` command. In that case the step size given in the `xaxis` command should be set small enough so that for each step the change in the error signal can be still approximated as an almost linear change. The table below shows the computing times for a Michelson `lock` in the GEO 600 input file. This very rough comparison of computation times shows that various step sizes work quite well (in this example the accuracy was set to 1 pW):

number of steps	700	800	900	1000	2000	3000	10000
computation time	61s	6s	5s	5s	6s	9s	28s

Please note that in case of the 700 steps FINESSE's autogain function has changed the loop gain automatically which also re-calibrates the step size. Even though in this case the lock was still succesful, it demonstrated that too large step sizes should be avoided. In the case of the 2000 steps, each step required about 22 internal iterations to reach the locking accuracy of 1 pW. An accuracy of 0.1 mW could be achieved with 5 iterations.

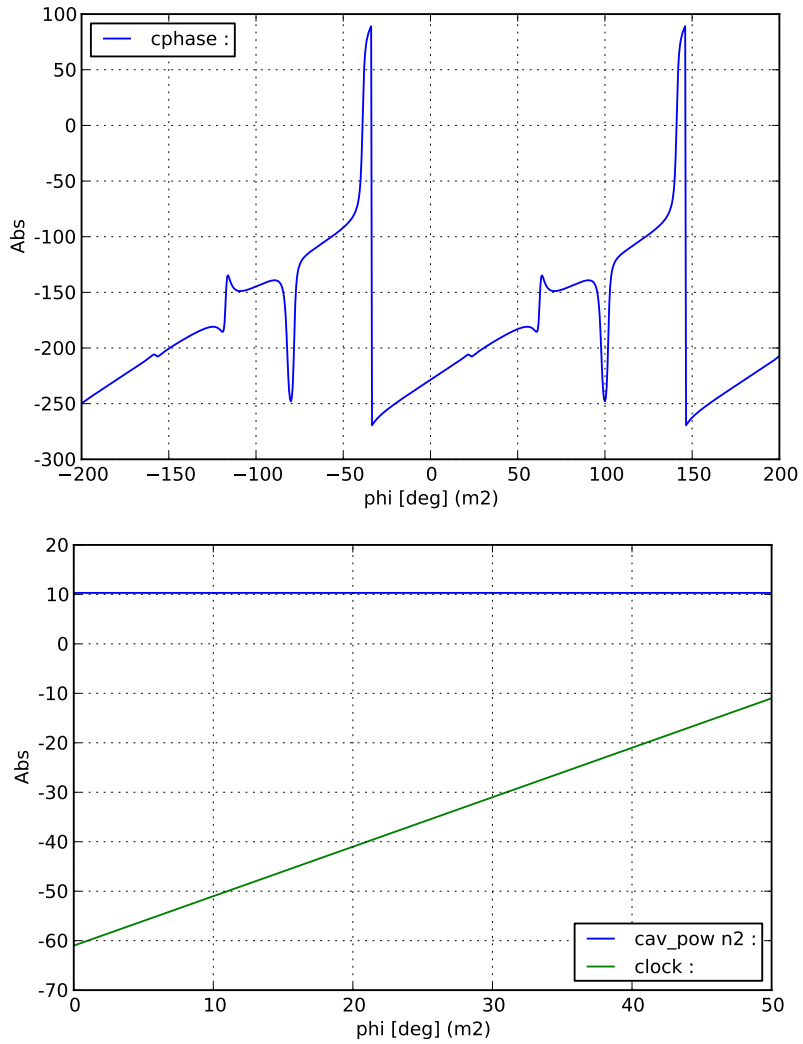


Figure 3.8: *Using a pseudo-lock signal for the* `lock` *command: the top plot shows the error signal, the lower plot shows the correction signal when the lock is active and the cavity circulating power as a test signal.*

3.5.4 A pseudo-lock

Another possibility to create a signal for the `lock` command is to use a so-called 'pseudo-lock' system in which the simulation signals, such as the phase of a light field is used to generate an error signal. Since such signal are not available in the experiment, this cannot represent a real control system. However, it has the advantage of generating an error signal without the need of extra optical components, suc as the electro-optical modulator for the PDH scheme shown above. An example for a pseudo-lock in shown in Figure 3.8 and the FINESSE file is given below.

It should be noted that pseudo-locks can be problematic, if they are not set up very carefully, their operating point might not be identical to that of the real control loop, which would cause the model to behave differently than the experiment and thus produce misleading and wrong results. This has been the reason behind a great many wasted modelling efforts in the GW community!

```
l i1 1 0 n0
gauss g1 i1 n0 22m -1.2
startnode n0

mod eo1 40k 0.3 3 pm n0 n1      # phase modulator f_mod=40kHz
                                # midx=0.3 order=3
m m1 0.95 0.01 $m1tune n1 n2    # cavity
s s2 1200 n2 n3
m m2 0.99 0.01 0 n3 n4
cav cav1 m1 n2 m2 n3

attr m2 Rc 3000                 # distortions, unmatched
attr m1 xbeta 1u                # curvature and misalignment
phase 0                         # turn off phase adjustement


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pseudo lock of cavity
ad ph_m11 0 n1*
ad ph_m12 0 n2*
noplot ph_m11
noplot ph_m12
set ph1 ph_m11 deg
set ph2 ph_m12 deg
set offset m1 phi
func cphase = unwrap($ph2 - 2*$offset) - $ph1 -90
noplot cphase
lock clock $cphase .02 1m
%noplot clock
put* m1 phi $clock
pd cav_pow n2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
const m1tune 100
xaxis m2 phi lin 0 50 800
maxtem 3

yaxis  abs
```

Chapter 4

Higher-order spatial modes, the paraxial approximation

The analysis using a plane-wave approximation as described in the previous chapter allows one to perform a large variety of simulations. Some analysis tasks, however, include the beam shape and position, i.e. the properties of the field transverse to the optical axis. The effects of misaligned components, for example, can only be computed if beam shape and position are taken into account.

The following sections describe a straightforward extension of the previous chapter's analysis using transverse electromagnetic modes (TEM). The expression *mode* in connection with laser light usually refers to the eigenmodes of a cavity. Here, one distinguishes between longitudinal modes (along the optical axis) and transverse modes, the spatial distribution of the light beam perpendicular to the optical axis. In the following, we are looking at the spatial properties of a laser beam. A *beam* in this sense is a light field for which the power is confined to a small volume around one axis (the optical axis, always denoted by z).

4.1 Finesse with Hermite-Gaussian beams

By default FINESSE performs simulations using the plane-wave approximation. If the input file contains commands which refer explicitly to Gaussian beams, like, for example, `gauss`, FINESSE will use Hermite-Gaussian beams instead. This is henceforth called the 'Hermite-Gauss extension'.

The command `maxtem` is used to switch manually between plane-waves and Gaussian beams and to set the maximum order for higher order TEM modes:
`maxtem off` switches to plane waves,
`maxtem order` with `order` a integer between 0 and 100 switches to Hermite-Gauss beams. The simulation includes higher order modes TEM_{nm} with $n + m \leq$`order`.

The Hermite-Gauss extension of FINESSE is a powerful tool. However, it requires some expert knowledge about the physics and its numerical representation. The following sections provide the mathematical description of the Gaussian beams as it is used in

FINESSE. Please see the extra section G.3 in the syntax reference for a description of commands relevant to Hermite-Gauss beams.

4.2 Gaussian beams

Imagine an electric field that can be described as a sum of the different frequency components and of the different spatial modes:

$$E(t,x,y,z) \;=\; \sum_j \sum_{n,m} a_{jnm}\; u_{nm}(x,y,z)\;\exp\left(\mathrm{i}\left(\omega_j\, t - k_j z\right)\right), \tag{4.1}$$

with u_{nm} describing the spatial properties of the beam and a_{jnm} as complex amplitude factors (ω_j is the angular frequency of the light field and $k_j = \omega_j/c$).

Please note that in this case the amplitude coefficients a_{jnm} are not equivalent to the field amplitudes as computed by FINESSE. There is a difference in phase as a consequence of the chosen implementation of the Gouy phase; see Section 4.3.3 for details. In the following the amplitudes a_{jnm} refer to the coefficients as defined in Equation 4.1. The amplitude coefficients computed and stored by FINESSE will be denoted b_{jnm}.

For simplicity we restrict the following description to a single frequency component at one moment in time ($t = 0$):

$$E(x,y,z) \;=\; \exp\left(-\mathrm{i}\,kz\right) \sum_{n,m} a_{nm}\; u_{nm}(x,y,z). \tag{4.2}$$

A useful mathematical model for describing spatial properties of light fields in laser interferometers are the Hermite-Gauss modes, which are the eigenmodes of a general spherical cavity (an optical cavity with spherical mirrors) and represent an exact solution of the *paraxial wave equation*; see Appendix F.2.

The following section provides an introduction to Hermite-Gauss modes, including some useful formulae and the description of the implementation of Hermite-Gauss modes in FINESSE. Please note that the paraxial approximation requires a well aligned and well mode-matched interferometer; Section 4.9 gives a short overview of the limits of this approximation.

The *Gaussian beam* often describes a simple laser beam to a good approximation. The Gaussian beam as such is the lowest-order Hermite-Gauss mode u_{00} which will be discussed later. The electric field (again assuming a single frequency and $t = 0$) is given as:

$$\begin{aligned} E(x,y,z) &= E_0 u_{00} \exp\left(-\mathrm{i}\,kz\right) \\ &= E_0 \left(\tfrac{1}{R_C(z)} - \mathrm{i}\,\tfrac{\lambda}{\pi w^2(z)}\right) \cdot \exp\left(-\mathrm{i}\,k\tfrac{x^2+y^2}{2R_C(z)} - \tfrac{x^2+y^2}{w^2(z)} - \mathrm{i}\,kz\right). \end{aligned} \tag{4.3}$$

The shape of a Gaussian beam is quite simple: the beam has a circular cross-section, and the radial intensity profile of a beam with total power P is given by:

$$I(r) = \frac{2P}{\pi w^2(z)} \exp\left(-2r^2/w^2\right), \tag{4.4}$$

with w the *spot size*, defined as the *radius* at which the intensity is $1/e^2$ times the maximum intensity $I(0)$. This is a Gaussian distribution, hence the name *Gaussian beam*. Figure 4.1 shows a cross-section through a Gaussian beam and the radial intensity for different positions with respect to the beam position and beam size.

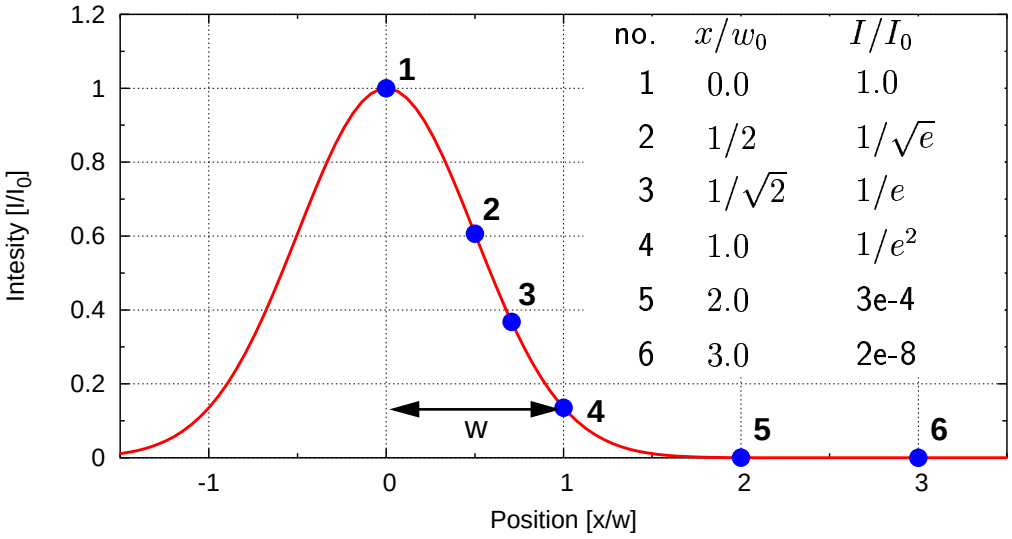


Figure 4.1: *One dimensional cross-section of a Gaussian beam. The width of the beam is given by the radius w at which the intensity is $1/e^2$ of the maximum intensity.*

Such a beam profile (for a beam with a given wavelength λ) can be completely determined by two parameters: the size of the minimum spot size w_0 (called *beam waist*) and the position z_0 of the beam waist along the z-axis.

To characterise a Gaussian beam, some useful parameters can be derived from w_0 and z_0. A Gaussian beam can be divided into two different sections along the z-axis: a *near field* (a region around the beam waist) and a *far field* (far away from the waist). The length of the near-field region is approximately given by the *Rayleigh range* z_R. The Rayleigh range and the spot size are related by the following expression:

$$z_\mathrm{R} = \frac{\pi w_0^2}{\lambda}. \tag{4.5}$$

With the Rayleigh range and the location of the beam waist, we can write the following useful expression:

$$w(z) = w_0 \sqrt{1 + \left(\frac{z - z_0}{z_\mathrm{R}}\right)^2}. \tag{4.6}$$

This equation gives the size of the beam along the z-axis. In the far-field regime ($z \gg z_\mathrm{R}, z_0$), it can be approximated by a linear equation:

$$w(z) \approx w_0 \frac{z}{z_\mathrm{R}} = \frac{z\lambda}{\pi\omega_0}. \tag{4.7}$$

The angle Θ between the z-axis and $w(z)$ in the far field is called the *diffraction angle*[1] and is defined as:

$$\Theta = \arctan\left(\frac{w_0}{z_\mathrm{R}}\right) = \arctan\left(\frac{\lambda}{\pi w_0}\right) \approx \frac{w_0}{z_\mathrm{R}}. \tag{4.8}$$

Another useful parameter is the *radius of curvature* of the wavefront at a given point z. The radius of curvature describes the curvature of the 'phase front' of the electromagnetic wave (a surface across the beam with equal phase) at the position z. We obtain for the radius of curvature as a fuction of z:

$$R_C(z) = z - z_0 + \frac{z_\mathrm{R}^2}{z - z_0}. \tag{4.9}$$

For the radius of curvature we also find:

$$\begin{aligned} &R_C \approx \infty, && z - z_0 \ll z_\mathrm{R} && \text{(beam waist)} \\ &R_C \approx z, && z \gg z_\mathrm{R},\ z_0 && \text{(far field)} \\ &R_C = 2z_\mathrm{R}, && z - z_0 = z_\mathrm{R} && \text{(maximum curvature)} \end{aligned} \tag{4.10}$$

4.3 Higher order Hermite-Gauss modes

The Hermite-Gauss modes are usually given in their orthonormal form as:

$$\begin{aligned} u_\mathrm{nm}(x,y,z) \quad &= \quad \left(2^{n+m-1} n! m! \pi\right)^{-1/2} \frac{1}{w(z)} \exp\left(\mathrm{i}\,(n+m+1)\Psi(z)\right) \\ &\times \quad H_n\left(\frac{\sqrt{2}x}{w(z)}\right) H_m\left(\frac{\sqrt{2}y}{w(z)}\right) \exp\left(-\mathrm{i}\,\frac{k(x^2+y^2)}{2R_C(z)} - \frac{x^2+y^2}{w^2(z)}\right), \end{aligned} \tag{4.11}$$

with n, m being the *mode numbers* or *mode indices*. In this case n refers to the modes in the y-z plane (saggital) and m to the x-z plane (tangential). The following functions are used in the equation above:

- $H_n(x)$: Hermite polynomial of the order n (unnormalised), see Appendix F.1,
- $w(z)$: beam radius or spot size,
- $R_\mathrm{C}(z)$: radius of curvature of the phase front,
- $\Psi(z)$: Gouy phase.

[1] Also known as the *far-field angle* or the *divergence* of the beam.

The definition of $\Psi(z)$ and some explanation is given in Section 4.3.3. The Hermite-Gauss modes can also be given in a very compact form using the Gaussian beam parameter q; see below.

The Hermite-Gauss modes as given above are orthonormal and thus:

$$\iint dxdy\ u_{nm}u^*_{n'm'} \ = \ \delta_{nn'}\delta_{mm'}. \tag{4.12}$$

Therefore the power of a beam, as given by Equation 4.2, being detected on a single-element photodetector (provided that the area of the detector is large with respect to the beam) can be computed as

$$P = \sum_{n,m} a_{nm}a^*_{nm}. \tag{4.13}$$

Or for a beam with several frequency components (compare with Equation 3.125):

$$P = \sum_{n,m}\sum_{i}\sum_{j} a_{inm}a^*_{jnm} \quad \text{with} \quad \{i,j \mid i,j \in \{0,\ldots,N\} \ \wedge \ \omega_i = \omega_j\}. \tag{4.14}$$

The x and y dependencies can be separated so that:

$$u_{nm}(x,y,z) \ = \ u_n(x,z)u_m(y,z). \tag{4.15}$$

4.3.1 Gaussian beam parameter

A set of Hermite-Gauss modes u_{nm} can be described by one constant beam parameter q_0: the *Gaussian beam parameter*. It is defined as:

$$\frac{1}{q(z)} = \frac{1}{R_C(z)} - \mathrm{i}\,\frac{\lambda}{\pi w^2(z)}, \tag{4.16}$$

and can also be written as:

$$q(z) = \mathrm{i}\,z_\mathrm{R} + z - z_0 = q_0 + z - z_0, \qquad \text{where} \qquad q_0 = \mathrm{i}\,z_\mathrm{R}. \tag{4.17}$$

The beam parameter q_0 is in general changed when the beam interacts with a spherical surface.

Using this parameter Equation 4.3 can be rewritten as:

$$u(x,y,z) = \frac{1}{q(z)}\exp\left(-\mathrm{i}\,k\frac{x^2+y^2}{2q(z)}\right). \tag{4.18}$$

The complete set of solutions as given in Equation 4.11 can now be written as[2]:

$$u_\mathrm{nm}(x,y,z) = u_\mathrm{n}(x,z)u_\mathrm{m}(y,z), \tag{4.19}$$

[2] Please note that this formula from [Siegman] is very compact. Since the parameter q is a complex number, the expression contains at least two complex square roots. The complex square root requires a different algebra than the standard square root for real numbers. Especially the third and fourth factors *cannot* be simplified in any obvious way i.e.: $\left(\frac{q_0}{q(z)}\right)^{1/2}\left(\frac{q_0 q^*(z)}{q_0^* q(z)}\right)^{n/2} \neq \left(\frac{q_0^{n+1}q^{*\,n}(z)}{q^{n+1}(z)q_0^{*\,n}}\right)^{1/2}$!

with

$$u_{\mathrm{n}}(x,z) = \left(\frac{2}{\pi}\right)^{1/4} \left(\frac{1}{2^n n! w_0}\right)^{1/2} \left(\frac{q_0}{q(z)}\right)^{1/2} \left(\frac{q_0\ q^*(z)}{q_0^*\ q(z)}\right)^{n/2} H_n\left(\frac{\sqrt{2}x}{w(z)}\right) \exp\left(-\mathrm{i}\,\frac{kx^2}{2q(z)}\right) \tag{4.20}$$

again, $H_n(x)$ represents a Hermite polynomial of order n.

The beam size and radius of curvature can also be written in terms of the beam parameter q:

$$w^2(z) = \frac{\lambda}{\pi}\frac{|q|^2}{\mathrm{Im}\,\{q\}}, \tag{4.21}$$

and

$$R_C(z) = \frac{|q|^2}{\mathrm{Re}\,\{q\}}. \tag{4.22}$$

It is clear that when using Hermite-Gauss modes one has to choose a base system of beam parameters for describing the spatial properties. In FINESSE this means a beam parameter has to be set for every node. Much of this task is automated; see Section 4.4. In my experience the quality of the simulations and the correctness of the results depend critically on the choice of these beam parameters. One might argue that the choice of the base system should not alter the result. This is correct but there is a practical limitation: the number of modes having non-neglible power might become very large if the beam parameters are not optimised, so that in practise to achieve a sensible computation time a good set of beam parameters must be used. Section 4.9 gives some advice how to ensure that the simulation does not fail because of this limitation.

4.3.2 Tangential and sagittal plane

If the interferometer is confined to a plane as in FINESSE, it is convenient to use projections of the three-dimensional description into two planes: the tangential plane, defined as the x-z plane and the sagittal plane as given by y and z.

The beam parameter can then be split into two beam parameters: q_{s} for the sagittal plane and q_{t} for the tangential plane so that the Hermite-Gauss modes can be written as:

$$u_{nm}(x,y) = u_n(x,q_{\mathrm{t}})\ u_m(y,q_{\mathrm{s}}). \tag{4.23}$$

Remember that these Hermite-Gauss modes form a base system. This means one can use the separation in sagittal and tangential planes even if the analysed optical system does not show this special type of asymmetry. This separation is very useful in simplifying the mathematics.

In the following, the term *beam parameter* generally refers to a simple q_0 but all the results can also be applied directly to a pair of parameters q_s, q_t.

4.3.3 Gouy phase shift

The introduction of spatial beam properties using Hermite-Gauss modes gives rise to an extra longitudinal phase lag, this is the *Gouy phase*. Compared to a plane wave, the Hermite-Gauss modes have a slightly slower phase velocity, especially close to the waist. The Gouy phase can be written as:

$$\Psi(z) = \arctan\left(\frac{z - z_0}{z_\mathrm{R}}\right), \tag{4.24}$$

or, using the Gaussian beam parameter:

$$\Psi(z) = \arctan\left(\frac{\mathrm{Re}\{q\}}{\mathrm{Im}\{q\}}\right). \tag{4.25}$$

Compared to a plane wave, the phase lag φ of a Hermite-Gauss mode is:

$$\varphi = (n + m + 1)\Psi(z). \tag{4.26}$$

With an astigmatic beam, i.e. different beam parameters in the tangential and saggital planes this becomes:

$$\varphi = \left(n + \frac{1}{2}\right)\Psi_t(z) + \left(m + \frac{1}{2}\right)\Psi_s(z), \tag{4.27}$$

with

$$\Psi_t(z) = \arctan\left(\frac{\mathrm{Re}\{q_t\}}{\mathrm{Im}\{q_t\}}\right), \tag{4.28}$$

as the Gouy phase in the tangential plane (and Ψ_s similarly the Gouy phase in the sagittal plane).

The command `phase` can be used to specify how the Gouy phase is used within the FINESSE simulation, see Section G.3.

4.3.4 ABCD matrices

The transformation of the beam parameter can be performed by the ABCD matrix-formalism [Siegman]. When a beam passes a mirror, beam splitter, lens or free space, a beam parameter q_1 is transformed to q_2. This transformation can be described by four real coefficients like so:

$$\frac{q_2}{n_2} = \frac{A\frac{q_1}{n_1} + B}{C\frac{q_1}{n_1} + D}, \tag{4.29}$$

with the coefficient matrix,

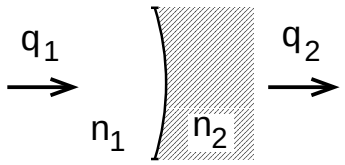
$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tag{4.30}$$

and n_1 being the index of refraction at the beam segment defined by q_1 and n_2 the index of refraction at the beam segment described by q_2.

The ABCD matrices for the optical components used by FINESSE are given below, for the sagittal and tangential plane respectively.

Transmission through a mirror: A mirror in this context is a single, partly reflecting surface with an angle of incidence of 90°. The transmission is described by:

$$M = \begin{pmatrix} 1 & 0 \\ \frac{n_2 - n_1}{R_C} & 1 \end{pmatrix} \tag{4.31}$$

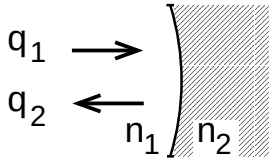


with R_C being the radius of curvature of the spherical surface. The sign of the radius is defined such that R_C is negative if the centre of the sphere is located in the direction of propagation. The curvature shown above (in Equation 4.31), for example, is described by a positive radius.

The matrix for the transmission in the opposite direction of propagation is identical.

Reflection at a mirror: The matrix for reflection is given by:

$$M = \begin{pmatrix} 1 & 0 \\ -\frac{2n_1}{R_C} & 1 \end{pmatrix} \tag{4.32}$$

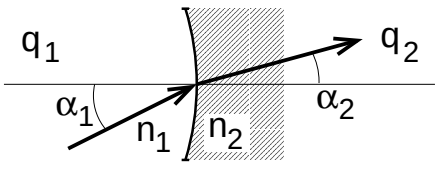


The reflection at the back surface can be described by the same type of matrix by setting $C = 2n_2/R_C$.

Transmission through a beam splitter: A beam splitter is understood as a single surface with an arbitrary angle of incidence α_1. The matrices for transmission and reflection are different for the sagittal and tangential planes (M_s and M_t):

$$M_t = \begin{pmatrix} \frac{\cos(\alpha_2)}{\cos(\alpha_1)} & 0 \\ \frac{\Delta n}{R_C} & \frac{\cos(\alpha_1)}{\cos(\alpha_2)} \end{pmatrix}$$

$$M_s = \begin{pmatrix} 1 & 0 \\ \frac{\Delta n}{R_C} & 1 \end{pmatrix} \tag{4.33}$$



with α_2 given by Snell's law:

$$n_1 \sin(\alpha_1) = n_2 \sin(\alpha_2), \tag{4.34}$$

and Δn for the tangential beam is:

$$\Delta n = \frac{n_2 \cos(\alpha_2) - n_1 \cos(\alpha_1)}{\cos(\alpha_1)\cos(\alpha_2)}. \tag{4.35}$$

and for the sagittal beam

$$\Delta n = n_2 \cos(\alpha_2) - n_1 \cos(\alpha_1) \tag{4.36}$$

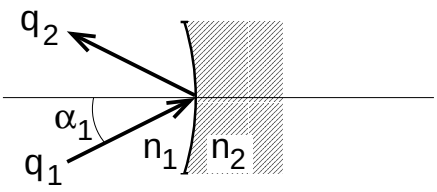
If the direction of propagation is reversed, the matrix for the sagittal plane is identical and the matrix for the tangential plane can be obtained by changing the coefficients A and D as follows:

$$\begin{aligned} A &\longrightarrow 1/A, \\ D &\longrightarrow 1/D. \end{aligned} \tag{4.37}$$

Reflection at a beam splitter: The reflection at the front surface of a beam splitter is given by:

$$M_{\mathrm{t}} = \begin{pmatrix} 1 & 0 \\ -\frac{2n_1}{R_{\mathrm{C}}\cos(\alpha_1)} & 1 \end{pmatrix}$$

$$M_{\mathrm{s}} = \begin{pmatrix} 1 & 0 \\ -\frac{2n_1\cos(\alpha_1)}{R_{\mathrm{C}}} & 1 \end{pmatrix} \tag{4.38}$$

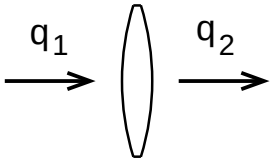


To describe a reflection at the back surface the matrices have to be changed as follows:

$$\begin{aligned} R_{\mathrm{C}} &\longrightarrow -R_{\mathrm{C}}, \\ n_1 &\longrightarrow n_2, \\ \alpha_1 &\longrightarrow -\alpha_2. \end{aligned} \tag{4.39}$$

Transmission through a thin lens: A thin lens transforms the beam parameter as follows:

$$M = \begin{pmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{pmatrix} \tag{4.40}$$



where f is the focal length. The matrix for the opposite direction of propagation is identical. Please note that a thin lens has to be surrounded by 'spaces' with index of refraction $n = 1$.

Transmission through a free space: As mentioned above, the beam in free space can be described by one base parameter q_0. In some cases it is convenient to use a similar matrix as for the other components to describe the z-dependency of $q(z) = q_0 + z$. On propagation through a free space of the length L and index of refraction n the beam parameter is transformed as follows:

$$M = \begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \qquad \text{q}_1 \longrightarrow \boxed{\text{Space}} \longrightarrow \text{q}_2 \tag{4.41}$$

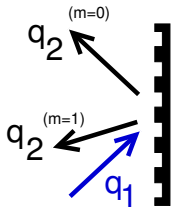
The matrix for the opposite direction of propagation is identical.

Reflection at a grating: Consider a curved diffraction grating with radius of curvature R, ruling in the y-direction and grating spacing d in the x-direction. An incident beam striking the grating at an angle α from the normal in the $x-z$ plane will be diffracted in mth order into angle ϕ_m in the same plane given by the grating equation Equation 3.55.

The matrix for reflection from the grating in the tangential or $x-z$ plane is given by:

$$M_{\text{t}} = \begin{pmatrix} M & 0 \\ -2/R_t & 1/M \end{pmatrix}$$

$$M_{\text{s}} = \begin{pmatrix} 1 & 0 \\ -2/R_s & 1 \end{pmatrix} \tag{4.42}$$



with M given by:

$$M = \cos(\phi_m)/\cos(\alpha), \tag{4.43}$$

ï¿½ and effective radii as:

$$R_t = R\,\frac{\cos(\alpha)\cos(\phi_m)}{\cos(\alpha)+\cos(\phi_m)}, \tag{4.44}$$

$$R_s = \frac{2\,R}{\cos(\alpha)+\cos(\phi_m)}. \tag{4.45}$$

4.4 Tracing the beam

As described in Section 4.3.1 one important step in the simulation is the choice of beam parameters. In general the Gaussian beam parameter of a TEM_{00} mode is changed at every optical surface (see Section 4.3.4). In other words, for each location inside the

interferometer where field amplitudes are to be computed a certain beam parameter has to be set for the simulation.

A possible method to find reasonable beam parameters for every location in the interferometer (every node in FINESSE) is to first set only some specific beam parameters and then derive the remaining beam parameters from these initial ones: usually it is sensible to assume that the beam at the input can be properly described by the (hopefully known) beam parameter of the laser's output mode. In addition, in most cavities the light fields can be described safely by using cavity eigenmodes. FINESSE provides two commands, `gauss` and `cav`, that can be used for setting beam parameters. The command `cav` computes the eigenmodes of a (stable) cavity and sets the respective beam parameters on every node that is part of the cavity. Whereas the command `gauss` is used to set a beam parameter at one specific node, for example, at the input. The two types of commands are executed as follows:

- Each `cav` command checks if the respective cavity is stable and, if so, it computes the eigenmode, and sets the respective beam parameters at every node that is inside the cavity.
- The `cav` commands are executed in the order they appear in the input file. If two cavities include the same node, the later command overwrites the beam parameter set for this node by previous `cav` commands.
- After the cavities have been all set, the `gauss` commands are executed. If a `gauss` command is set to a node inside a cavity the beam parameter set by the `cav` command will be overwritten by the one given in the `gauss` command.

After some beam parameters have been set by `gauss` or `cav` commands FINESSE uses a beam tracing algorithm to set beam parameters for the remaining nodes. 'Trace' in this context means that a beam starting at a node with an already known beam parameter is propagated through the optical system and the beam parameter is transformed according to the optical elements encountered.

The tracing algorithm in FINESSE works as follows:

- the starting point of the tracing can be set explicitly by the user with the command `startnode nodename` (the beam parameter of the respective node has to be set with either `gauss` or `cav`). If this command is not used the starting point is automatically set:
  - to the input node of the (first) cavity if the user has specified at least one stable cavity;
  - to the node given in the first `gauss` command, if the user specified at least one Gaussian parameter but no cavity;
  - to the node of the first laser if the user did not specify any beam parameter. In addition, FINESSE sets the beam parameter of that input node to a default beam parameter: $q = \mathrm{i}\,(2\,\mathrm{mm})^2\pi/\lambda$ (i.e. $w_0 = 2\,\mathrm{mm}$, $z_0 = 0\,\mathrm{mm}$).
- from the starting point the beam is traced through the full interferometer simply by following all possible paths successively. This is done by moving from the start node to a connecting component then to the next node, to the next component and

so on. At every optical element along the path the beam parameter is transformed according to the ABCD matrix of the element (see below). If more than one possibility exists (for example at a beam splitter) the various paths are followed one after the other. Each path ends, i.e. is considered to be traced completely, when an already encountered node or a 'dump' node is found.

- Every time a new node is found for which no beam parameter has been yet set the current beam parameter is set for that node.
- If a new node is found that already has a beam parameter (for example, by the user with a `gauss` command) the current beam parameter is dropped and the parameter of the node is kept and used for further tracing along that path instead. In this case the tracing algorithm makes sure that no such beam parameter change occurs inside space components[3].
- When all paths have been traced completely, the number of nodes found is compared to the total number of nodes of the setup and an error is generated if these numbers do not match.

During the simulation, if a length, radius of curvature, or focal length is changed, the optimum set of base parameters changes. When FINESSE detects a change in one of these parameters is automatically recompute the best beam parameters for each data point. This will slow down the simulation a little but in almost all cases it yields much better results. You can use the command `retrace` to force FINESSE to recompute beam parameters for each data point. Or you can force it to switch retracing off in all cases, using the command `retrace off`.

FINESSE can provide plenty of information about the tracing and the resulting beam parameters. The command `trace` can be used to set the verbosity of FINESSE's tracing algorithm or, more generally, the verbosity of the Hermite-Gauss mode; see the table on page 210 in the syntax reference.

4.5 Interferometer matrix with Hermite-Gauss modes

In the plane-wave analysis, a laser beam was described in general by the sum of various frequency components of its electric field:

$$E(t,z) \;=\; \sum_j a_j \;\exp\Big(\mathrm{i}\,(\omega_j\,t - k_j z)\Big). \tag{4.46}$$

Now, the geometric shape of the beam is included by describing each frequency component by a sum of Hermite-Gauss modes:

$$E(t,x,y,z) \;=\; \sum_j \sum_{n,m} a_{jnm}\; u_{nm}(x,y)\;\exp\left(\mathrm{i}\,(\omega_j\,t - k_j z)\right). \tag{4.47}$$

[3] It is important for the chosen implementation of the Gouy phase (see Section 4.5) that the beam parameter for both nodes of a space component refer to the same beam waist.

The shape of such a beam does not change along the z-axis (in the paraxial approximation). More precisely, the spot size and the position of the maximum intensity with respect to the z-axis may change, but the relative intensity distribution across the beam does not change its shape.

Each part of the sum may be treated as an independent field that can be described using the equation for the plane-wave approximation with only two exceptions:

- the propagation through free space has to include the Gouy phase shift, and
- upon reflection or transmission at a mirror or beam splitter the different Hermite-Gauss modes may be coupled (see below).

The Gouy phase shift can be included into the simulation in several ways. For reasons of flexibility is has been included in FINESSE as a phase shift of the component space. The beam trace algorithm has been designed to set the beam parameters of a space component so that at both nodes the beam parameter gives the same Gouy phases. Therefore it is possible to associate the component with a known phase delay. The amplitude of a field propagating through a space is thus given by:

$$b_{\text{out}} = b_{\text{in}} \exp\left(\mathrm{i}\,\Delta\omega n_r L/c - \left(\frac{1}{2}+n\right)\Psi_x + \left(\frac{1}{2}+m\right)\Psi_y\right), \tag{4.48}$$

(compare to Equation 3.40).

This means the Gouy phases are stored explicitly in the amplitude coefficients. Therefore, the amplitudes $b_{\text{in/out}}$ are not equivalent to these a_{jnm} in Equation 4.47 or Equation 4.1. In fact, the field amplitude is given in FINESSE (for one point in space, and $t = 0$) as:

$$E(t,x,y,z) = \sum_j \sum_{n,m} b_{jnm}\, u_n(x) u_m(y) \exp\left(-\mathrm{i}\left(\frac{1}{2}+n\right)\Psi_t)\right) \exp\left(-\mathrm{i}\left(\frac{1}{2}+m\right)\Psi_s)\right), \tag{4.49}$$

with

$$\Psi_t = \arctan\left(\frac{\mathrm{Re}\,\{q_t\}}{\mathrm{Im}\,\{q_t\}}\right), \qquad \Psi_s = \arctan\left(\frac{\mathrm{Re}\,\{q_s\}}{\mathrm{Im}\,\{q_s\}}\right). \tag{4.50}$$

This formula is used, for example, with the `beam` detector.

Also, changing from one TEM base system to another it is necessary to turn back the Gouy phase with respect to the old beam parameter and add the Gouy phase with respect to the new beam parameter. This is required because the coupling coefficients used in the computation in Section 4.6.1 were derived from the field description given by Equation 4.1 for which the Gouy phase is not stored in the amplitude coefficients but implicitly given by the spatial distribution.

4.6 Coupling of Hermite-Gauss modes

The following is based on the work of F. Bayer-Helms [Bayer-Helms]. I later discovered that there exists a very good description of coupling coefficients by J. Y. Vinet [VPB].

Let us assume two different cavities with different sets of eigenmodes. The first set is characterised by the beam parameter q_1 and the second by the parameter q_2. A beam with all power in the fundamental mode $\mathrm{TEM}_{00}(q_1)$ leaves the first cavity and is injected into the second. Here, two 'mis-configurations' are possible:

- if the optical axes of the beam and the second cavity do not overlap perfectly, the setup is called *misaligned*,
- if the beam size or shape at the second cavity does not match the beam shape and size of the (resonant) fundamental eigenmode ($q_1(z_{\mathrm{cav}}) \neq q_2(z_{\mathrm{cav}})$), the beam is then not *mode-matched* to the second cavity, i.e. there is a *mode mismatch*.

The above mis-configurations can be used in the context of simple beam segments. In the simulation, the beam parameter for the input light is specified by the user. Ideally, the ABCD matrices allow one to trace a beam through the optical system by computing the proper beam parameter for each beam segment. In this case, the basis system of Hermite-Gauss modes is transformed in the same way as the beam so that the modes are *not coupled*.

For example, an input beam described by the beam parameter q_1 is passed through several optical components, and at each component the beam parameter is transformed according to the respective ABCD matrix. Thus, the electric field in each beam segment is described by Hermite-Gauss modes based on different beam parameters, but the relative power between the Hermite-Gauss modes with different mode numbers remains constant, i.e. a beam in a TEM_{00} mode is described as a pure TEM_{00} mode throughout the full system.

In practice, it is usually impossible to compute proper beam parameters for each beam segment as above, especially when the beam passes a certain segment more than once. The most simple example is the reflection at a spherical mirror. Let the input beam be described by q_1. From Equation 4.32 we know that the proper beam parameter of the reflected beam is:

$$q_2 = \frac{q_1}{-2q_1/R_{\mathrm{C}} + 1}, \tag{4.51}$$

with R_{C} being the radius of curvature of the mirror. In general, we get $q_1 \neq q_2$ and thus two different 'proper' beam parameters for the same beam segment. Only one special radius of curvature would result in matched beam parameters ($q_1 = q_2$).

4.6.1 Coupling coefficients for TEM modes

The Hermite-Gauss modes are coupled whenever a beam is not matched to a cavity or to a beam segment or if the beam and the segment are misaligned. In this case, the beam has

to be described using the parameters of the beam segment (beam parameter and optical axis). This is always possible (provided that the paraxial approximation holds) because each set of Hermite-Gauss modes (defined by the beam parameter at a position z) forms a complete set. Such a change of the basis system results in a different distribution of light power in the (new) Hermite-Gauss modes and can be expressed by coupling coefficients that yield the change in the light amplitude and phase with respect to mode number.

Let us assume a beam described by the beam parameter q_1 being injected into a segment described by the parameter q_2. Let the optical axis of the beam be misaligned: the coordinate system of the beam is given by (x, y, z) and the beam travels along the z-axis. The beam segment is parallel to the z'-axis and the coordinate system (x', y', z') is given by rotating the (x, y, z) system around the y-axis by the *misalignment angle* γ. The coupling coefficients are defined as:

$$u_{nm}(q_1) \exp\Big(\mathrm{i}\,(\omega t - kz)\Big) = \sum_{n',m'} k_{n,m,n',m'} u_{n'm'}(q_2) \exp\Big(\mathrm{i}\,(\omega t - kz')\Big), \tag{4.52}$$

where $u_{nm}(q_1)$ are the Hermite-Gauss modes used to describe the injected beam and $u_{n'm'}(q_2)$ are the 'new' modes that are used to describe the light in the beam segment. Please note that including the plane wave phase propagation into the definition of coupling coefficients is very important because it results in coupling coefficients that are independent of the position on the optical axis for which the coupling coefficients are computed.

Using the fact that the Hermite-Gauss modes u_{nm} are orthonormal, we can compute the coupling coefficients by the following inner product [Bayer-Helms]:

$$k_{n,m,n',m'} = \exp\Big(\mathrm{i}\,2kz' \sin^2\Big(\frac{\gamma}{2}\Big)\Big) \iint dx'dy'\ u_{n'm'} \exp\big(\mathrm{i}\,kx' \sin\gamma\big)\ u^*_{nm} \tag{4.53}$$

$$= \exp\Big(\mathrm{i}\,2kz' \sin^2\Big(\frac{\gamma}{2}\Big)\Big) \big\langle u_{n'm'} \exp\big(\mathrm{i}\,kx' \sin\gamma\big), u_{nm}\big\rangle\,. \tag{4.54}$$

Since the Hermite-Gauss modes can be separated with respect to x and y, the coupling coefficents can also be split into $k_{nmn'm'} = k_{nn'}k_{mm'}$. These equations are very useful in the paraxial approximation as the coupling coefficients decrease with large mode numbers. In order to be described as paraxial, the angle γ must not be larger than the diffraction angle. In addition, to obtain correct results with a finite number of modes the beam parameters q_1 and q_2 must not differ too much, see Section 4.9.

The convolution given in Equation 4.53 can be directly computed using numerical integration. This is computationally very expensive. In [Bayer-Helms] the above projection integral is partly solved and the coupling coefficients are given by simple sums as functions of γ and the mode mismatch parameter K, which are defined by:

$$K = \frac{1}{2}(K_0 + \mathrm{i}\,K_2), \tag{4.55}$$

where $K_0 = (z_R - z'_R)/z'_R$ and $K_2 = ((z - z_0) - (z' - z'_0))/z'_R$. This can be also written

as (using $q = \mathrm{i}\, z_\mathrm{R} + z - z_0$):

$$K = \frac{\mathrm{i}\,(q - q')^*}{2\,\mathrm{Im}(q')}. \tag{4.56}$$

The coupling coefficients for misalignment and mismatch (but no lateral displacement) can be then be written as:

$$k_{nn'} = (-1)^{n'} E^{(x)} (n!n'!)^{1/2} (1+K_0)^{n/2+1/4} (1+K^*)^{-(n+n'+1)/2} \left\{ S_g - S_u \right\}, \tag{4.57}$$

where:

$$\begin{aligned} S_g &= \sum_{\mu=0}^{[n/2]} \sum_{\mu'=0}^{[n'/2]} \frac{(-1)^\mu \bar{X}^{n-2\mu} X^{n'-2\mu'}}{(n-2\mu)!(n'-2\mu')!} \sum_{\sigma=0}^{\min(\mu,\mu')} \frac{(-1)^\sigma \bar{F}^{\mu-\sigma} F^{\mu'-\sigma}}{(2\sigma)!(\mu-\sigma)!(\mu'-\sigma)!}, \\ S_u &= \sum_{\mu=0}^{[(n-1)/2]} \sum_{\mu'=0}^{[(n'-1)/2]} \frac{(-1)^\mu \bar{X}^{n-2\mu-1} X^{n'-2\mu'-1}}{(n-2\mu-1)!(n'-2\mu'-1)!} \sum_{\sigma=0}^{\min(\mu,\mu')} \frac{(-1)^\sigma \bar{F}^{\mu-\sigma} F^{\mu'-\sigma}}{(2\sigma+1)!(\mu-\sigma)!(\mu'-\sigma)!}. \end{aligned} \tag{4.58}$$

S_u does not exist for $n = n' = 0$, due to negative factorials in the denominator. The respective formula for $k_{mm'}$ can be obtained by replacing the following parameters: $n \to m$, $n' \to m'$, $X, \bar{X} \to 0$ and $E^{(x)} \to 1$ (see below). The notation $[n/2]$ means:

$$\left[\frac{m}{2}\right] = \begin{cases} m/2 & \text{if } m \text{ is even,} \\ (m-1)/2 & \text{if } m \text{ is odd.} \end{cases} \tag{4.59}$$

The other abbreviations used in the above definition are:

$$\begin{aligned} \bar{X} &= (\mathrm{i}\, z'_\mathrm{R} - z') \sin(\gamma) / (\sqrt{1+K^*} w_0), \\ X &= (\mathrm{i}\, z_\mathrm{R} + z') \sin(\gamma) / (\sqrt{1+K^*} w_0), \\ F &= K/(2(1+K_0)), \\ \bar{F} &= K^*/2, \\ E^{(x)} &= \exp\left(-\tfrac{X\bar{X}}{2}\right). \end{aligned} \tag{4.60}$$

In general, the Gaussian beam parameter might be different for the sagittal and tangential planes and a misalignment can be given for both possible axes (around the y-axis and around the x-axis), in this case the coupling coefficients are given by:

$$k_{nmm'n'} = k_{nn'} k_{mm'}, \tag{4.61}$$

where $k_{nn'}$ is given above with

$$\begin{aligned} &q \to q_t \\ &\text{and} \\ &w_0 \to w_{t,0}, \text{ etc.}, \end{aligned} \tag{4.62}$$

and $\gamma \to \gamma_y$ is a rotation about the y-axis. The $k_{mm'}$ can be obtained with the same formula, replacing:

$$\begin{aligned} &n \to m, \\ &n' \to m', \\ &q \to q_s, \\ &\text{thus} \\ &w_0 \to w_{s,0}, \text{ etc.}, \end{aligned} \tag{4.63}$$

and $\gamma \to \gamma_x$ is a rotation about the x-axis.

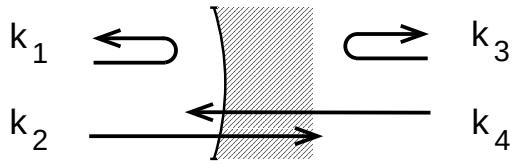


Figure 4.2: *Coupling coefficients for Hermite-Gauss modes: for each optical element and each direction of propagation complex coefficients k for transmission and reflection have to be computed. In this figure k_1, k_2, k_3, k_4 each represents a matrix of coefficients $k_{nmn'm'}$ describing the coupling of* $\mathrm{TEM}_{n,m}$ *into* $\mathrm{TEM}_{n',m'}$.

At each component a matrix of coupling coefficients has to be computed for transmission and reflection; see Figure 4.2.

4.6.2 Alignment transfer function

An alignment transfer function in this context is the ratio between any interferometer output signal and a periodic change of the alignment angle of a mirror or beam splitter. Such transfer functions are needed for calculating the coupling of alignment noise into longitudinal interferometer signals and for the design of auto-alignment control systems.

As in the plane-wave approximation the source signal is injected into the interferometer via the command `fsig`. A modulation of the alignment of a mirror or beam splitter creates sidebands at the modulation frequency. We now derive the amplitude and phase of these sidebands (with respect to the impinging light).

First, the amplitude coefficients a_{jnm} of the incoming light field are stored, with j as the frequency index and n, m as the TEM mode indices. Then the static couplings between TEM modes due to a possible static misalignment or mode mismatch are computed, so that the amplitude coefficients of the reflected field can be computed:

$$a'_{jn'm'} = \sum_{n,m} k_{n'm'nm} a_{jnm} \exp\left(\mathrm{i}\,\varphi_{\mathrm{sb}}\right), \tag{4.64}$$

where φ_{sb} is the plane wave phase shift acquired through a possible detuning of the component from the reference plane as given by Equation 3.103.

The beam, now given by $a'_{jn'm'}$ is again misaligned by an angle of $\gamma = \epsilon \sin(\omega_m t)$. The coupling coefficients simplify if only misalignment is considered:

$$k_{nn'} = (-1)^{n'} E^{(x)} (n!n'!)^{1/2} \sum_{\mu=0}^{\min(n,n')} \frac{(-1)^\mu \bar{X}^{n-\mu} X^{n'-\mu}}{\mu!(n-\mu)!(n'-\mu)!}, \tag{4.65}$$

where

$$\bar{X} = X^* = \frac{-(z' - \mathrm{i}\, z'_{\mathrm{R}}) \sin\gamma}{w'_0}, \tag{4.66}$$

For the transfer function, only the terms linear in γ and thus linear in X have to be considered. Only one addend of the above sum can be linear in X and only if $|n-n'| = 1$. In edition the exponential function can be neglected $E^{(y)} = 1, E^{(x)} = 1 + O(\gamma^2)$. Thus the sum reduces to:

$$\sum_{\mu=0}^{\min(n,n')} \frac{(-1)^\mu \bar{X}^{n-\mu} X^{n'-\mu}}{\mu!(n-\mu)!(n'-\mu)!} = O(X^2) + \begin{cases} \frac{(-1)^{n'}\bar{X}}{n'!} & \text{for} \quad n = n'+1 \\ \frac{(-1)^n X}{n!} & \text{for} \quad n' = n+1 \\ 1 & \text{for} \quad n' = n \\ 0 & \text{otherwise} \end{cases} \tag{4.67}$$

The coupling coefficients can now be written as:

$$k_{nn'} = \begin{cases} \sqrt{n}\bar{X} & \text{for} \quad n = n'+1, \\ -\sqrt{n'}\bar{X}^* & \text{for} \quad n' = n+1, \\ 1 & \text{for} \quad n' = n, \\ 0 & \text{otherwise.} \end{cases} \tag{4.68}$$

4.7 Mirror surface maps

The term *mirror map* often refers to a scan of a manufactured mirror obtained by interferometric means, the resulting map contains, for example, measurements of the surface height or substrate transmission measured at the nodes of an x, y grid. More generally we can think of a *surface map* as a two dimensional data array describing properties of the reflection or transmission of an optical surface as a function of the position on the surface.

Typically a mirror map is subject to pre-processing to remove average effects, like a tilt of the entire surface arising from the measurement process. Often, further processing is done to remove the expected optical profile of the mirror in order to measure only the residual deviations, for example the surface roughness.

Phase maps, for example, are of great importance for the purpose of estimating noises introduced by mirror surface aberrations. In the initial design of an interferometer we assume the mirrors to have perfectly smooth surfaces, yet the manufacturing process introduces various surface distortions, for example, a surface roughness on the order of a

nanometre. By providing an accurate simulation including surface distortions, potential design problems can be identified before the experimental apparatus is being build.

Three types of surface maps can be applied to `mirror` components in FINESSE: *phase maps*, *absorption maps* and *reflectivity maps*. 'phase maps' and 'absorption maps' can be set to affect only the reflected light, only the transmitted light or affecting both light fields. 'Reflectivity maps' always change the reflected and transmitted light. Please note that the name 'phase map' refers to the effect the map has on the impinging light field. However, the numerical values stored in the map files are not phases but surface distortions in meters. The values for higher parts of the surface are defined as positive whereas 'holes' are indicated by negative values. The numerical data in 'absorption maps' represent power loss coefficients (values between 0 and 1) and 'reflectivity maps' are composed of power reflectivity coefficients (values between 0 and 1).

If the amplitude reflectivity r of a mirror is constant over its entire surface but the mirror surface is not perfectly smooth and the surface data is available, the reflection of a field from a mirror can be described by a phase map. Alternatively we can imagine mirrors which are considered perfectly smooth but are subject to absorption that varies with the exact position on the surface. This effect can be studied using absorption maps. Real mirrors are never perfectly smooth, nor do they feature a perfectly homogeneous reflectance or transmittance. Therefore, in general a mirror surface could be best described with a map storing phase and amplitude information together. However, the implementation of surface maps in FINESSE uses separate maps, giving information either on the effects on the light phase, the absorption or the reflectivity. Yet, several maps of different types can be applied to the same surface simultaneously.

4.7.1 Phase maps

The effect of a phase map on the reflected field can be described mathematically as:

$$E_{\text{refl}} \;=\; r \;\exp\left(\mathrm{i}\,\varphi(x,y)\right)\; E_{\text{in}} \tag{4.69}$$

with

$$\varphi(x,y) = 2\frac{n_1\omega}{\lambda\,\omega_0}\phi_p(x,y) \tag{4.70}$$

and n_1 the index of refraction of the medium we are in and ϕ_p the measured surface map of the mirror surface, given in metres. Positive values refer to higher parts of the surface and negative to lower.

The frequencies are defined as usual, ω is the frequency of the light field impinging on the surface and ω_0 the frequency corresponding to the default wavelength λ_0. For simplicity we assume that λ can be replaced by λ_0 and the factor ω/ω_0 can be approximated as 1. We then obtain:

$$E_{\text{refl}} \;=\; r \;\exp\left(2\mathrm{i}\; n_1\,\phi_p(x,y)/\lambda_0\right)\; E_{\text{in}} \tag{4.71}$$

The transmission can be written as

$$E_{\text{trans}} = \text{i}\ t\ \exp\left(\text{i}\ (n_1 - n_2)\,\phi_p(x,y)/\lambda_0\right)\ E_{\text{in}} \tag{4.72}$$

4.7.2 Absorption maps

The reflected field subject to an absorption map can be written as

$$E_{\text{refl}} = r\ \sqrt{1 - L(x,y)}\ E_{\text{in}} \tag{4.73}$$

with $L(x, zy)$ the measured absorption map of the mirror surface, given in power coefficients. The transmission can be written as

$$E_{\text{trans}} = \text{i}\ t\ \sqrt{1 - L(x,y)}\ E_{\text{in}} \tag{4.74}$$

4.7.3 Reflectivity maps

Reflectivity maps are implemented slightly differently because they replace the mirror parameters r and t given in the `mirror` command. These values are used to compute a loss factor $L = 1 - r^2 - t^2$ and then are set internally to 1. The reflected field can then be written as

$$E_{\text{refl}} = \sqrt{R(x,y)}\ \sqrt{1 - L}\ E_{\text{in}} \tag{4.75}$$

with $R(x, zy)$ the measured reflectivity map of the mirror surface, given in power coefficients. The transmitted field can be written as

$$E_{\text{trans}} = \text{i}\ \sqrt{1 - R(x,y)}\ \sqrt{1 - L}\ E_{\text{in}} \tag{4.76}$$

4.7.4 Coupling coefficients from mirror maps

In FINESSE the shape of a field is not given as a function of x,y coordinates but by a sum of Hermite-Gauss modes of different orders. Therefore the effect of a mirror map needs to be described as scattering into higher order modes. The coupling coefficients can be computed by the usual integral. In the case of the reflection we obtain, for example:

$$k_{n,m,n',m'} = \int\int dx'\,dy'\ u_{n',m'}\ \exp(2\text{i}\ n_1\,\phi_p(x',y')/\lambda_0)\,u^*_{n,m} \tag{4.77}$$

which can be computed directly using a numerical integration routine or in some specific cases analytically.

In order to be compatible with already existing coupling coefficients the following approach has been chosen: The coupling coefficients due to a mirror map are computed independently of any misalignment or change in Gaussian beam parameter that occurs

at the given mirror. Therefore for any given map function $A(x, y)$ the coefficients with respect to a field impinging on the front face of the mirror are computed as:

$$k^{\text{map}}_{n,m,n',m'} = \int\int dx'\, dy'\; u(n', m', q_1, n_1)\, A(x', y')\, u^*(n, m, q_1, n_1) \tag{4.78}$$

with q_1 and n_1 being the Gaussian beam parameter and the index of refraction of the node in front of the mirror, respectively. This equation is true for reflection as well as transmission (the map function A would be different between those cases of course). The coefficients are then merged with the other coupling coefficients, see Appendix E. This approach allows to seamlessly use together multiple maps as well as additional attributes to mirrors such as radius of curvature and alignment angle. However, the sperate computation and subsequent merging of coupling coefficients is an approximation when only a finite set of modes is used. In consequence, one has to be very careful in setting up a model using maps; there are a few configuration commands which can be used to select the best method for computing and merging the coefficients, see Section 4.7.13.

The following table[4] gives an overview of the map functions in the different cases. $B(x, y)$ shall be a matrix of real numbers representing the data stored in the map file.

type of map	*fields affected*	$A(x,y)$ *(refl.)*	$A(x,y)$ *(trans.)*
phase	reflection	$\exp(\mathrm{i}\, 2k\, n_1 B(x,y))$	1
phase	transmisson	1	$\exp(-\mathrm{i}\, kB(x,y))$
phase	both	$\exp(\mathrm{i}\, 2k\, n_1 B(x,y))$	$\exp(-\mathrm{i}\, k\,(n_1 - n_2)B(x,y))$
absorption	reflection	$\sqrt{1 - B(x,y)}$	1
absorption	transmission	1	$\sqrt{1 - B(x,y)}$
absorption	both	$\sqrt{1 - B(x,y)}$	$\sqrt{1 - B(x,y)}$
reflectivity	both	$\sqrt{1 - L}\sqrt{B(x,y)}$	$\sqrt{1 - L}\sqrt{1 - B(x,y)}$

with n_1, n_2 the indices of refraction of the medium before and after the surface respectively.

4.7.5 The map file format

A mirror map file can contains the mirror map as a grid $B(x, y)$. The data grid $B(x, y)$ must be stored as follows. The data is preceded by a header consisting of seven lines, for example:

```
% Surface map
% Name: test
% Type: phase transmission
% Size: 201 201
% Optical center (x,y): 101 101
% Step size (x,y): 0.0001 0.0001
% Scaling: 5.32e-07
```

[4] Please note in the different entries of the table $B((x, y)$ represents different data types of different dimension.

The first line indicates that a map of grid data follows, the second line specifies the name of the map and the third line the type of the map. Possible types are:

- phase transmission
- phase reflection
- phase both
- absorption transmission
- absorption reflection
- absorption both
- reflectivity both

The fourth line states the number of rows and columns of the map data, line five gives the optical center (in real numbers referring to grid indices, starting at zero), typically the center is at (cols+1)/2,(rows+1)/2.

Line number six gives the physical length (in meters) of one grid elements in the x and y directions of one grid element. The overall size of the grid in relation to the size of the light field must be chosen very carefully to avoid numerical errors. FINESSE computes a typical size of the light field as follows. The maximum diameter in the horizontal direction is given as:

$$d_{\mathrm{beam,x}} = 2\ w_x(z)\ \sqrt{\mathrm{maxtem} + 0.5} \tag{4.79}$$

The maximum beam diameter is then computed as

$$d_{\mathrm{beam}} = max(d_{\mathrm{beam,x}}, d_{\mathrm{beam,y}}) \tag{4.80}$$

FINESSE then computes an effective size of the grid as two times the smallest distance from the optical center to the edge. FINESSE issues a warning if the such computed grid size is smaller than four times the maximum beam size.

The last line of the header defines a scaling factor to be applied to the data that follows.

This header is then followed by the grid data stored in columns and rows as given by the 'Size' in the header. The grid can contain four different kinds of information specified by type of the map (see list above). Phase maps store information related to optical path length, given in meters, amplitude related maps store power coefficients between 0 and 1. The grid data is then used inside FINESSE to compute coupling coefficients as given by Equation 4.78. Note that in all cases the Gaussian beam parameter used to compute $u(n', m')$ and $u^*(n, m)$ are identical.

4.7.6 How to apply a map to a component

Applying a map to a component means you have one of the maps types discussed above already at your disposal. Take the script snippet below, we have a mirror which we want to apply one or several maps to, this is simply done using the `map` command.

```
m m1 0.99 0.01 0 n2 n3

# map command usage: map [component_name] [map_filename]
map m1 aperture_map.txt
map m1 surface_roughness_map.txt
map m1 reflectivity_map.txt
```

Above we have applied 3 maps to our mirror - the maximum is 30 though using that many is unlikely - all three maps must have exactly the same physical sizes and discretisations. In previous versions of Finesse each map would have a separate matrix computed of coupling coefficients, which were later matrix multiplied together. This provides an interesting problem as matrix multiplication is not commutative, so the result will differ depending on the order in which the maps are specified in the kat file. To get around this issue the maps are merged together to form one single merged-map. This is done by representing the maps in complex exponential form and each of the maps multiplied together. This merged-map is what is used to compute the coupling coefficients.

4.7.7 Accelerating calculations by saving coupling coefficients

The process of calculation the coupling coefficient integral can be incredibly slow by nature. If you have a simulation which requires some coupling coefficients and you need to run the simulation multiple times you can save the coupling coefficients to a file. This is done using the `knm` command:

```
m m1 0.99 0.01 0 n2 n3
map m1 aperture_map.txt
map m1 reflectivity_map.txt

# knm usage: knm [component_name] [filename_prefix]
knm m1 test1_m1
```

The above will save the coupling coefficients generated by the various maps you apply to a file called `test1_m1.knm`. It will also save 4 other files which save the merged-map in amplitude and phase components for reflection and transmission called `.map` files. It is important that these 5 files be kept together if you want to reuse the cofficients.

The `knm` command not only saves the cofficients but also tells Finesse to load the files aswell, the `filename_prefix` argument just needs to be the same as what you saved with, without the filename extension `.knm` though.

Finesse will then try to load the 5 files. The conditions in which the saved coefficients were calculated are stored in the `.knm` file. If the saved conditions are different to what the simulation is now trying to run, the saved files will be ignored and a new set of coefficients will be generated and saved. In the `.knm` file you will also notice lines like `map0 : mymap_aperture.txt PWPU75F7XZIGxURwdEiJIA==`, which list the maps that have been applied to the mirror. The random looking string is infact a hash that is uniquely generated to the contents of map file `mymap_aperture.txt`. If anything in that file changes the calculation will be redone. Also it is important to note that reordering

the maps in the kat file will also cause the computations to be recalculated. The hash is an MD5 hash that when written to the files is stored using a BASE64 conversion.

The 5 files however are not hash protected, thus if you change the files by hand in a text editor you can load the files using the `knm` command and the computation will be done with any changes you have made - this functionality may change in later versions as of `0.99.9`.

The integration routine can also be sped up by using the symmetric nature of the coupling coefficient matrix in certain conditions. This is detailed further in appendix E.3. This speeds up the computation by $\approx \times 2$ as only the upper half and diagonal elements of the matrix need computing, the lower half can be inferred from the upper. This can be switched on or off from the `kat.ini` file by using the option `calc_knm_tranpose 0 (off) or 1 (on)`. This is switched on by default.

4.7.8 Coupling cofficient data files - ASCII vs binary formats

When using particularly large maps or a large `maxtem` value you will quickly find that reading and writing the 5 map files becomes painfully slow. To combat this we added the ability to save the files in either ASCII (Normal readable text) or binary (unreadable) format. Changing which format is used is done using the `conf` command per component:

```
conf [component name] save_knm_binary [0 (ASCII) or 1 (Binary)]
```

If you try to load a binary or ASCII file when Finesse has been told to use the other, an error will occur.

4.7.9 Integration and interpolation methods

The integral is performed numerically using an integrating library named Cuba or by a simplistic Riemann sum. The Riemann sum works by summing over all grid elements, the x, y coordinates used in the Hermite-Gauss functions u_{nm} are computed as follows:

$$\begin{aligned} x(i = 1 : cols) &= (i - x0) * xstep \\ y(j = 1 : rows) &= (j - y0) * ystep \end{aligned} \tag{4.81}$$

with $cols, rows$ the number of columns and rows in the data grid, $x0$ and $y0$ the indices of the optical center as given in the header and $xstep, ystep$ the lengths of one grid element. If a rotation angle is given in the `map` command, x and y are further rotated by minus the given angle (If no angle is given and no angle has been found in the file, an angle of zero degrees is assumed).

The Cuba integration routines are much more sophisticated in there approach in solving the coupling coefficient integrals. Essentially the routines sample the mirror map in a sparse fashion in an attempt to calculate the integral. The integration of the map is split into several domains, each domain has the integral calculated and an error estimated,

if the error is larger than the acceptable values the domain is further split and more samples are used. This way the routines should concentrate on areas of difficulty with more evaluations, whereas simple domains only require a few and are much more efficient.

As the routines sample the discretised map in a continuous manner, interpolation is needed to sample within the maps data points. The interpolation is provided by the GSL library and provides 3 options. The fastest method is Nearest-Neighbour, ideal if the map is sampled highly or you just want a quick integration. Linear interpolation, and the slowest, cubic interpolation is also offered. The choice of interpolation routine will depend on what data your map has been generated from. If the underlying data was flat/linear then linear interpolation would be ideal, however if original surface or data was smoothed, like a mirror surface, cubic might give a better representation of the real surface. It is important to remember that cubic interpolation can produce artefacts that may not exist due to the polynomials it forces to fit the points of the map, which is problematic for 'rough' maps. It is therefore recommended that you use linear interpolation for most purposes. If you are using a binary map, i.e. a mask of 1's and 0's, you must use nearest-neighbour to avoid varying values at points inbetween a 1 and 0.

There are several commands used to set the various integration and interpolation routines depending on your required use. First, you can specify the default integration and interpolation methods in the 'kat.ini' file so as to apply to multiple kat files. This is done by adding lines:

```
mapintmethod 1
mapinterpmethod 1
```

The various methods are chosen by a number 1, 2 or 3, for integration methods:

- Riemann Sum - 1
- Cuba Serial - 2
- Cuba Parallel -3

and for interpolation methods:

- Nearest Neighbour - 1
- Linear - 2
- Cubic -3

To set the default integration method for all components in a kat file you can use the `intmethod [integration_method]` command in your FINESSE input file.

You can also specify integration and interpolation on a component by component basis. This is in the case where one map might have a rough surface which requires Cuba integration but another very smooth and linear which could be done with a Riemann sum. This is done using the `conf` (configure) command, for as example as shown below:

```
map m2 mymap_aperture.txt # Here we apply our map to the mirror
conf m2 interpolation_method 1 # This sets the interpolation to nearest neighbour
conf m2 integration_method 1 # This sets the integration method to the Riemann sum
```

4.7.10 Map example: a focusing surface in transmission

This and the following example of using a map compare the results achieved with a surface map to an equivalent FINESSE result without a map. In this case we compare the focussing of beam by a lens (the built-in FINESSE command) to the focussing by a mirror map representing the same focal length.

The transmission map representing a lens has been created using a set of SimTools functions [SimTools], see Section 7.2.1, the MATLAB file is:

```
focallength = 200;
L1=10;
L2=150;
lambda = 1064e-9;
realsize = 0.7;
finesse_map_filename='lens_map.txt';
gridsize = 200;
map = FT_create_lens_map(gridsize,realsize,realsize,focallength);
FT_write_surface_map(finesse_map_filename,map);
```

To run a FINESSE simulation using this map, the following input file can be used:

```
const L1 10
const L2 150
const f 200

l i1 1 0 n1
gauss g1 i1 n1 1e-2 0

s s1 $L1 n1 n2
m m1 0 1 0 n2 n3
s s2 $L2 n3 n4

% load map file
map m1 lens_map.txt
% coefficients should be saved in files named 'lens_test.*'
knm m1 lens_test
% coefficients should be saved in binary format
conf m1 save_knm_binary 1
% integration method: Cuba serial
conf m1 integration_method 2
% interpolation method: linear
conf m1 interpolation_method 2
% change of Gaussian parameters computed in: map
conf m1 knm_change_q 2

% we explicitly set the beam parameter at the second
% node of the mirror to match the focussed beam, so that
% all the power should remain in the 00 mode
```

```
gauss g2 m1 n3 5.6961815m -138.35168

beam b1 n4
xaxis b1 x lin -1 1 50
trace 8
maxtem 2
phase 2
```

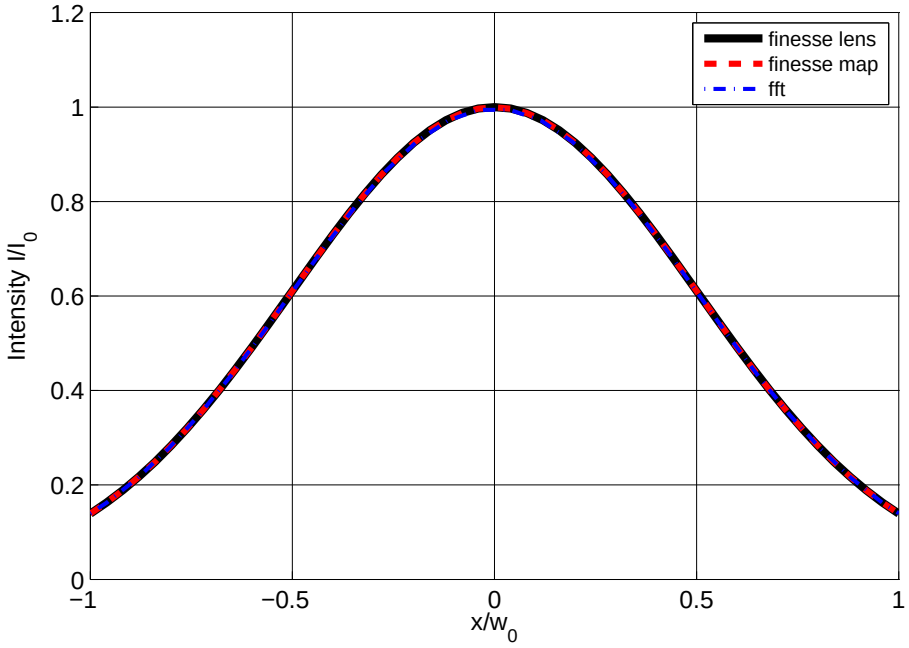


Figure 4.3: *Beam shape comparison between setup with a lens component and a mirror surface map representing the same focal length. Also shown is a result from an FFT propagation code using the same transmission map*

To check this result we can run a different file which uses a lens component instead of the mirror with the surface map:

```
const L1 10
const L2 150
const f 200

l i1 1 0 n1
gauss g1 i1 n1 1e-2 0

s s1 $L1 n1 n2
lens l1 $f n2 n3
s s2 $L2 n3 n4
gauss g2 l1 n3 5.6961815m -138.35168

beam b1 n4
```

```
xaxis b1 x lin -1 1 50
trace 8
maxtem 2
phase 2
```

Both results are shown in figure 4.3.

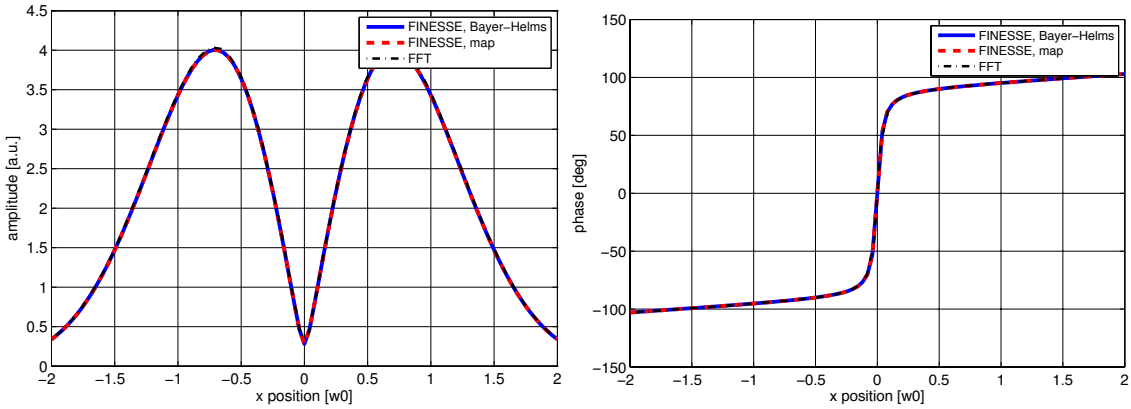


Figure 4.4: *Comparison of different simulations of the light reflected by a tilted mirror: a TEM_{00} mode is reflected by a mirror which is tilted by 1 µrad around the vertical axis. In order to highlight the effect of the tilt the TEM_{00} part of the reflected beam is removed so that only the TEM_{10} part remains. The above plots show a very good agreement between an FFT propagation with* FINESSE *results for using a mirror map or the built-in* `xbeta` *command.*

4.7.11 Surface map example: a tilted mirror in reflection

Another simple example is the tilted mirror: a beam in a TEM_{00} mode is reflected by a mirror. This mirror can be tilted, either using the `xbeta` command or by applying a tilted phase map to the mirror. The results are for both types of simulation and an equivalent FFT propagation are shown in Figure 4.4. In order to highlight the effect of the tilt, the TEM_{00} part of the beam has been removed so that only the TEM_{10} field shows. Both the amplitude and the phase distribution match precisely for all three methods.

The MATLAB script to generate the tile map is again based on SimTools and look as follows:

```
gridsize = 256;
% Full map radius and physical size of grid
R = 10;
realsize = 0.15;
% Tilt of 1urad along x-axis
xbeta = 1e-6;
ybeta = 0;
% Write tilt map
```

```
map = FT_create_tilted_map(xbeta,ybeta,R,realsize,gridsize);
FT_write_surface_map('tilted_map.txt',map)
```

The FINESSE input file is:

```
l i1 1 0 n1
gauss g1 i1 n1 20.3666m -10

s s1 10 n1 n2
m m1 0.99 0.01 0 n2 n3

map m1 tilted_map.txt
knm m1 tilt_test
conf m1 save_knm_binary 1
conf m1 integration_method 3
conf m1 interpolation_method 2

beam b1 0 n2
mask b1 0 0 0

maxtem 8
phase 2

xaxis b1 x lin -2 2 100
yaxis abs:deg
```

4.7.12 Realistic map example: thermal distortions

A more complicated example and one which takes care to simulate correctly is a cavity including thermal distortions of the mirrors. The high circulating powers in advanced gravitational wave detectors will lead to relatively large distortions of the mirrors making up the arm cavities. Simulation of such a setup requires careful calculation and preparation of the mirror maps describing the thermal distortions, taking care to remove any curvature and offset present in the mirrors and only including such effects in the FINESSE file. Simulations such as these are crucial for the commissioning of second generation gravitational wave detectors and, as such, are a good test of FINESSE as a robust and useful tool. An investigation into the round-trip losses incurred in a cavity with thermal distortions is described in section C, where we show that the correct result can be achieved with a high enough `maxtem`.

4.7.13 Couling coefficients for multiple effects

The separate computation of coupling coefficients of different effects at the same surface requires some care in setting up the model, in particular one must take care configuring the Gaussian beam parameters correctly.

Order of calculation

FINESSE can compute three different sets of coupling coefficients:

- analytic coefficients using the Bayer-Helms (BH) equations, which includes the effects from all parameters set with the `attr` command, such as misalignment and curvature.
- coefficients for mirror maps, computed through numerical integration
- coefficients for apertures, computed using numerical integration

In principle the order in which these are computed should not matter, however, with a finite number of modes the merging of these coefficients remains an approximation, and thus the order in which the coefficient matrices are merged can change the result. The magnitude of this change might serve as an indicator of the overall error due to the approximation of this approach. See appendix E for more mathematical details on the separation of the coupling coefficients.

The setup of a FINESSE model requires care only if a mirror map contains a residual curvature or astigmatism. The beam tracing algorithm in FINESSE does not know about the maps and thus choses sub-optimal beam parameters. Therefore we strongly recommend to remove any curvature and astigmatism from a map (using SimTools, see Section 7.2.1) and apply these instead using the `attr` command.

However, there might be special cases in which a curvature cannot be removed from a map. In this case we need to understand how FINESSE separates the coupling coefficient calculation into multiple matrices to speed up the calculation of static and dynamic effects, e.g. surface maps and tilts using the `xbeta` attribute (See appendix E). The coupling coefficient that includes both the effects of the surface map calculated by numerical integration and misalignments and mode-mismatches analytically with Bayer-Helms can be broken down into a matrix multiplication. Matrix K_A and K_B can represent either the results of numerical integration or Bayer-Helms. Here N and M represent the incoming and outgoing mode $n'm'$ and nm,

$$K_{NM} = [K_A K_B]_{NM} \tag{4.82}$$

$$= \sum_L^{\infty} \underbrace{\left\langle U_N(q_1')A(x,y), U_L(q_L)\right\rangle}_{\text{A Solver}} \underbrace{\left\langle U_L(q_L), B^*(x,y)U_M(q_2)\right\rangle}_{\text{B Solver}}. \tag{4.83}$$

The decision which needs to be made is what the value of q_L is. Reasonable choices are the incoming beam parameter q_1' and the outgoing beam parameter q_2. q_L should be chosen so that the solver that contains the mode-mismatch (whether that be a curved map or

Bayer-Helms with `attr`) has the incoming and outgoing q values in its inner product. This can be set using the `conf` command

```
conf [component name] knm_change_q [1 (for q'_1) or 2 (for q_2)]
```

To choose whether K_A or K_B represents the map integration or Bayer-Helms solver we use the command

```
conf [component name] knm_order [21 (K_A = Map, K_B = Bayer-Helms) or 12]
```

The default values are

```
knm_change_q 1
knm_order 21
```

The ordering is not overly important, from testing we have found the differences are minor as the number of modes is increased. However for ppm level or computation it might be of interest to swap the ordering to see if any commutation errors exist between the matrices. Depending on the order the different solvers will see different q values depending on the choice of q_L.

Choice of Gaussian beam parameter for multiple effects

If we repeat the example shown in Section 4.7.10 using a non-optimal configuration for the Gaussian beam parameters that are used by the numerical integration routine, we get a substantially different result. The FINESSE input file is:

```
const L1 10
const L2 150
const f 200

l i1 1 0 n1
gauss g1 i1 n1 1e-2 0

s s1 $L1 n1 n2
m m1 0 1 0 n2 n3
s s2 $L2 n3 n4

map m1 lens_map.txt
knm m1 lens_test
conf m1 save_knm_binary 1
conf m1 integration_method 2
conf m1 interpolation_method 2
conf m1 knm_change_q 1
gauss g2 m1 n3 5.6961815m -138.35168

maxtem 2
phase 2
beam b1 n4
```

```
xaxis b1 x lin -1 1 50
```

This is the same as before except for the line

```
conf m1 knm_change_q 1
```

The results are shown in Figure 4.5. Whereas the previous example gave correct results already at `maxtem 2`, in this example the correct result can only be achieved using `maxtem`>20! The reason for this is the mode mismatch at the lens, created by using an non-optimal selection of Gaussian parameters: The mirror map acting as a lens will transform the beam such that it the outgoing fundamental mode is described by a different Gaussian parameters than the incoming beam. By setting `conf m1 knm_change_q 1` however, the map coefficient calculation is forced to use the beam parameter of the incoming beam also for the outgoing field. The difference is substantial, the appropriate waist size for the outgoing beam would be $w_0 = 5.6961815\,\mathrm{mm}$ but here the coefficients are computed based on $w_0 = 10\,\mathrm{mm}$. In Section 4.9 we discuss the limits of the paraxial approximation on which the FINESSE algorithms are based, which includes the statement that the beam waist sizes should not differ by more than a factor of 3. While this example is therefore still within the limit of the paraxial approximation, it requires a relatively high number of modes to compute correct results.

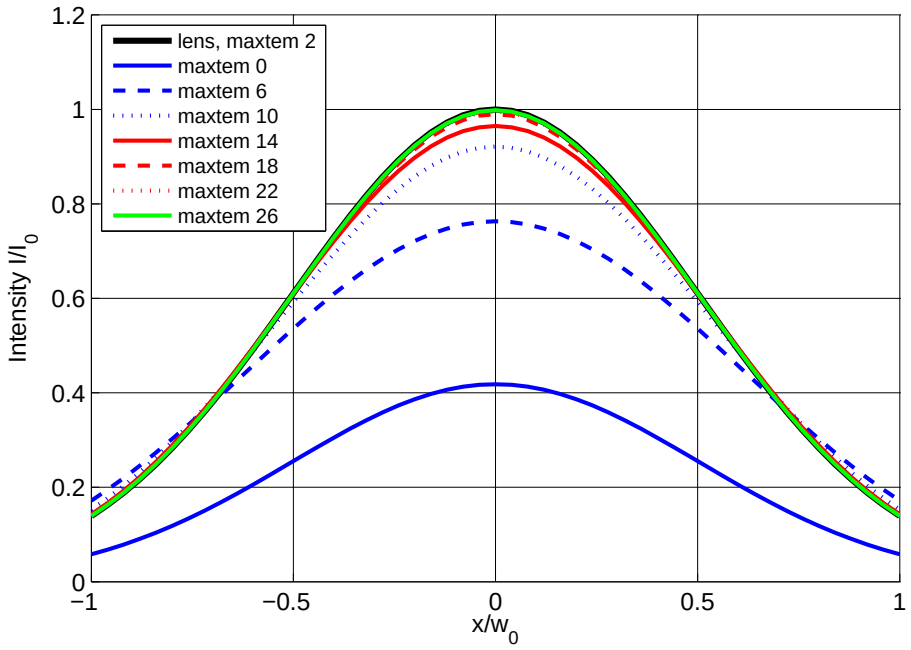


Figure 4.5: *Beam shape comparison between setup with a lens component and a mirror surface map representing the same focal length. The solid black trace shows the correct result obtained with the lens command. The other traces show the results for a mirror map for different maxtem, using a non-optimal configuration for the Gaussian beam parameter. Compare this to Figure 4.3, which uses a better configuration.*

You must make sure that if a map changes the beam parameter (i.e. it represents a curved surface), this beam parameter change is reflected in the Gaussian beam parameters provided to the integration routine computing the coupling coefficient; the `conf knm_change_q` command can be used for this purpose. However, we recommend to precondition maps such that all curvature are removed and are instead entered in the model via the radius of curvature parameters of the mirror element. By default this will create correct results with the least number of modes.

4.8 Detection of Hermite-Gauss modes

The Hermite-Gauss modes affect all detector types. The following sections describe the changes with respect to the plane-wave mode.

4.8.1 Amplitude detectors

The amplitude detectors in the Hermite-Gauss mode can be specified as

```
ad name n m f node
```

with n and m as the mode indices. Such a detector will plot the complex amplitude of each field with frequency f and mode indices n, m.

If the amplitude detector is used without specifying n and m, the detector tries to measure something like the phase front on the optical axis. To do so it computes the average of the powers in all modes at frequency f and computes the phase of the sum of these fields. The result can be written as follows:

$$S = a \exp(\mathrm{i}\,\Phi), \tag{4.84}$$

where

$$a = \sqrt{\sum_{n,m} |a_{n,m}|^2}, \tag{4.85}$$

$$\Phi = \mathrm{phase}\left(\sum_{n,m} a_{n,m}\right). \tag{4.86}$$

This feature has been tested to give similar values as a FFT propagation code on the optical axis. However, this feature should be considered as experimental.

4.8.2 Photodetectors

Since the Hermite-Gauss modes (as they are used here) are orthonormal, the photocurrent upon detection on a single-element photodiode (for simplicity shown here for one

frequency component only) is proportional to:

$$S = \sum_{n,m} a_{nm} a^*_{nm}. \tag{4.87}$$

(as usual the fields referring to sidebands created by `fsig` commands are ignored for the computation of the light power).

More interesting for the Hermite-Gauss modes are different detector types that are sensitive to the shape of the beam, for instance: split photodetectors. FINESSE can be used with such detectors of arbitrary design by defining the *beat coefficients*. For an arbitrary split detector the photocurrent is computed as:

$$S = \sum_{n,m} \sum_{n',m'} b_{nmn'm'} a_{nm} a^*_{n'm'}, \tag{4.88}$$

where b is the beat coefficient matrix. The beat coefficients in the equation 4.88 can be specified by the user through the `pdtype` command and the corresponding definition in the 'kat.ini' file. The subsequent demodulation of the signal is performed exactly as in plane-wave mode. In the following section we show an exemplary calculation of these coefficients for a split photodetector.

4.8.3 Split photodetector

A split photodetector is defined as a infinitely large plane perpendicular to the beam axis, split into two halves along one axis. The output signal is computed as the difference between the signals generated from each of the two sides. Here we consider the x-split photodetector, which is divided along the y-axis. To compute the coefficients $b_{nmn'm'}$ we assume a beam consisting of two different Hermite-Gauss modes is impinging on the detector. The separability of Hermite-Gauss modes in x and y allows us to separate the beat coefficients:

$$b_{nmn'm'} = b_{nn'} b_{mm'}. \tag{4.89}$$

For a detector split in the x-direction it is straightforward to find the beat coefficient factor $c_{mm'}$ corresponding to the the vertical mode indices. The orthonormality of the Hermite-Gauss functions means that:

$$b_{mm'} = \begin{cases} 1 \text{ when } m = m', \\ 0 \text{ when } m \neq m', \end{cases} \tag{4.90}$$

leading to a simplification of the full beat coefficients:

$$b_{nmn'm'} = \begin{cases} b_{nn'} \text{ when } m = m', \\ 0 \text{ when } m \neq m'. \end{cases} \tag{4.91}$$

The problem is now reduced to finding $b_{nn'}$, and as a result it is only necessary to consider the fields in the x-direction. The impinging field can now be written in one dimension as:

$$E(x) = a_n u_n(x) + a_{n'} u_{n'}(x). \tag{4.92}$$

The intensity becomes

$$I(x) = |E|^2 = a_n u_n a^*_{n'} u^*_{n'} + a^*_n u^*_n a_{n'} u_{n'} + |a_n u_n|^2 + |a_{n'} u_{n'}|^2 = I_1 + I_0 \tag{4.93}$$

with $I_0 = |a_n u_n|^2 + |a_{n'} u_{n'}|^2$.

The signal of the x-split detector is given by:

$$S = \int_0^\infty dx I(x) - \int_{-\infty}^0 dx I(x) = \int_0^\infty dx I(x) + \int_0^{-\infty} dx I(x) \tag{4.94}$$

$$= \int_0^\infty dx I(x) + \int_0^{-\infty} dx I(x) = \int_0^\infty dx I_1(x) - \int_0^\infty dx I_1(-x). \tag{4.95}$$

The I_0 contributions are cancelled from the signal because $I_0(-x) = I_0(x)$. Now we want to have a closer look at I_1. First we can show that $u_n u^*_{n'}$ is a real number. From equation 4.11 we get:

$$u_n u^*_{n'} = \sqrt{\frac{2}{\pi}} \sqrt{\frac{1}{2^{n+n'} n! n'! w^2}} e^{\mathrm{i}\,(n-n')\Psi} H_n\left(\frac{\sqrt{2}x}{w}\right) H_{n'}\left(\frac{\sqrt{2}x}{w}\right) e^{-\frac{2x^2}{w^2}}. \tag{4.96}$$

However in FINESSE the Gouy phase is stored in the field amplitudes a_n, so that with that in mind we calculate the following:

$$f_{nn'} = (u_n u^*_{n'})_{|\text{no Gouy phase}} = \sqrt{\frac{2}{\pi}} \sqrt{\frac{1}{2^{n+n'} n! n'! w^2}} H_n\left(\frac{\sqrt{2}x}{w}\right) H_{n'}\left(\frac{\sqrt{2}x}{w}\right) e^{-\frac{2x^2}{w^2}} \tag{4.97}$$

which is a real number. Thus we can write:

$$I_1 = a_n u_n a^*_{n'} u^*_{n'} + a^*_n u^*_n a_{n'} u_{n'} = (a_n a^*_{n'} + a^*_n a_{n'})\, f_{nn'} \tag{4.98}$$

and the split detector signal becomes

$$S = (a_n a^*_{n'} + a^*_n a_{n'}) \left(\int_0^\infty dx f_{nn'}(x) - \int_0^\infty dx f_{nn'}(-x) \right) = (a_n a^*_{n'} + a^*_n a_{n'})\, b_{nn'}. \tag{4.99}$$

From the definition of the Hermite polynomials in Appendix F.1 we can conclude that $f_{nn'}$ is an odd function when $n + n'$ is odd and an even function when $n + n'$ is even. Thus we get

$$b_{nn'} = \begin{cases} 0 & \text{for} \quad n + n' \text{ even}, \\ 2 \int_0^\infty dx f_{nn'}(x) & \text{for} \quad n + n' \text{ odd}. \end{cases} \tag{4.100}$$

Next we would like to find an analytical solution for the integral $\int_0^\infty dx f_{nn'}(x)$.

$$\int_0^\infty dx\, f_{nn'}(x) = \int_0^\infty dx \sqrt{\frac{2}{\pi}}\sqrt{\frac{1}{2^{n+n'}n!n'!w^2}} H_n\left(\frac{\sqrt{2}x}{w}\right) H_{n'}\left(\frac{\sqrt{2}x}{w}\right) e^{-\frac{2x^2}{w^2}} \tag{4.101}$$

$$= \sqrt{\frac{2}{\pi}}\sqrt{\frac{1}{2^{n+n'}n!n'!w^2}} \int_0^\infty dx\, H_n\left(\frac{\sqrt{2}x}{w}\right) H_{n'}\left(\frac{\sqrt{2}x}{w}\right) e^{-\frac{2x^2}{w^2}} \tag{4.102}$$

$$= \sqrt{\frac{2}{\pi}}\sqrt{\frac{1}{2^{n+n'}n!n'!w^2}}\; B_1 \tag{4.103}$$

The integral B_1 can be simplified by applying a variable substitution:

$$v = \frac{\sqrt{2}x}{w} \tag{4.104}$$

and we obtain:

$$B_1 = \frac{w}{\sqrt{2}} \int_0^\infty dv\, H_n(v)\, H_{n'}(v)\, e^{-v^2}. \tag{4.105}$$

To solve this we require the following two useful identities. For n being an odd number we get:

$$\int_0^\infty dx\, x^n e^{-x^2} = \frac{1}{2}\left(\frac{n-1}{2}\right)! \tag{4.106}$$

A Hermite polynomial can be written [Abramowitz] as:

$$H_n(x) = n! \sum_{l=0}^{\left[\frac{n}{2}\right]} (-1)^l \frac{1}{l!(n-2l)!}(2x)^{n-2l} \tag{4.107}$$

with

$$\left[\frac{n}{2}\right] = \begin{cases} \frac{n}{2} & \text{for} \quad n \text{ even}, \\ \frac{n-1}{2} & \text{for} \quad n \text{ odd}. \end{cases} \tag{4.108}$$

We require $n + n'$ to be odd, thus only n or n' can be odd. Assuming n to be even and n' to be odd we can write:

$$B_1 = \frac{w}{\sqrt{2}} \int_0^\infty du\, H_n(u)\, H_n'(u)\, e^{-u^2} \tag{4.109}$$

$$= \frac{w}{\sqrt{2}} n!n'! \sum_{l}^{n/2} \sum_{l'}^{(n'-1)/2} (-1)^l (-1)^{l'} \frac{1}{l!(n-2l)!} \frac{1}{l'!(n'-2l')!} 2^{n-2l} 2^{n'-2l'} \tag{4.110}$$

$$\cdot \int_0^\infty dx\, x^{n+n'-2l-2l'} e^{-x^2} \tag{4.111}$$

Knowing that $(n + n' - 2l - 2l')$ is an odd number we can replace the integral with the following:

$$\int_0^\infty dx\, x^{n+n'-2l-2l'} e^{-x^2} = \frac{1}{2}\left(\frac{n+n'-1}{2} - l - l'\right)! \tag{4.112}$$

and thus we obtain:

$$B_1 = \frac{w}{\sqrt{2}} n!n'!2^{n+n'-1} \sum_{l}^{n/2} \sum_{l'}^{(n'-1)/2} (-1)^{l+l'} 2^{-2l-2l'} \frac{\left(\frac{n+n'-1}{2} - l - l'\right)!}{l!l'!(n-2l)!(n'-2l')!} \tag{4.113}$$

and as our final result:

$$b_{nn'} = 2\sqrt{\frac{2}{\pi}}\sqrt{\frac{1}{2^{n+n'}n!n'!w^2}} B_1 \tag{4.114}$$

$$= \sqrt{\frac{2^{n+n'}n!n'!}{\pi}} \sum_{l}^{n/2} \sum_{l'}^{(n'-1)/2} \left(-\frac{1}{4}\right)^{l+l'} \frac{\left(\frac{n+n'-1}{2} - l - l'\right)!}{l!l'!(n-2l)!(n'-2l')!}. \tag{4.115}$$

By using split detectors in FINESSE, one may calculate the control signals for automatic alignment systems or other similar geometrical control systems.

The `kat.ini` file distributed with FINESSE contains the beat coefficients as described above for modes up to `maxtem 40`. These coefficients have been created using a SimTools script.

4.8.4 Beam detectors

The beam detector has two modes. If the command is used without specifying a frequency it acts like a CCD camera, it plots the beam intensity as a function of the x and y coordinate perpendicular to the optical axis. The output is a real number computed as:

$$s(x,y) = \sum_{ij}\sum_{nm} u_{nm}(x,y)u^*_{nm}(x,y)a_{inm}a^*_{jnm} \quad \text{with} \quad \{i,j \mid i,j \in \{0,\ldots,N\} \wedge \omega_i = \omega_j\}. \tag{4.116}$$

If instead a frequency is specified the beam detector resembles an amplitude detector, it outputs the amplitude and the phase of a the light field at the given frequency as a function of the x and y coordinate. The light field at frequency ω_i is given by a complex number (z), and is calculated as follows:

$$z(x,y) = \sum_{j}\sum_{nm} u_{nm}(x,y)a_{jnm} \quad \text{with} \quad \{j \mid j \in \{0,\ldots,N\} \ \wedge \ \omega_j = \omega_i\}. \tag{4.117}$$

4.9 Limits to the paraxial approximation

The decomposition of a laser beam into a set of Hermite-Gauss modes is merely an approximation. Also, the coupling coefficients as given in [Bayer-Helms] are derived using additional approximations. In order to obtain sensible results one has to understand the limits of these approximations. From references given within [Bayer-Helms] the following simple criteria can be determined. The paraxial approximation can be understood as a first order approximation in the parameter κ with:

$$\kappa = \left(\frac{w_0}{2z_\mathrm{R}}\right)^2 = \left(\frac{\lambda}{2\pi w_0}\right)^2. \tag{4.118}$$

In general we can assume that the approximation is valid for $\kappa \ll 1$ and the error will be of the order of κ^2. In the case of coupling one beam into another, the two characteristic parameters should be of the same order of magnitude.

In order to calculate some limits the above criteria are translated into:

- $\kappa < 0.1$
- $0.1 < \kappa_1/\kappa_2 < 10$

From the limit on κ one can directly derive that the divergence angle of the beam should be approximately less than 35°, which corresponds to limits computed in [Siegman]. From the limit on the relative difference of κ_1 and κ_2 one can derive that the waist size of the two beams should not differ by more than a factor of $\sqrt{10}$. Also assuming that the beam size should never exceed these limits, we can calculate that the waist position should not differ by more than three times the (smaller) Rayleigh range.

In conclusion, we believe that the following criteria can be used as a rough guide to judge whether the computation stays within the limits of the relevant approximations:

- the diffraction angle of every beam should be less than 30°;
- any misalignment between two beams should not be larger than their diffraction angles;
- the waist sizes of the beams should not differ by more than a factor of three;
- the distance between the waist positions of the beams should be smaller than three times the Rayleigh ranges;

Please note that the above limits do *not* imply that correct results can be reached by using a reasonable number of modes. In practice, much stronger limits have to be applied to reach acceptable computation times; see below.

In summary, for a perfectly aligned and mode-matched interferometer the results will be correct. Both misalignment and mode mismatch (or not optimally chosen Gaussian beam parameters) result in light being transferred into higher-order modes. In general, the number of modes that have to be taken into account depends on the amount of the misalignment or the amount of mode mismatch.

4.10 Mode mismatch in practice when using Finesse

Mode matching effects can complicate any simulation of interferometer layouts using higher order modes. A mode mismatch in this context refers to any interference between two beams which are best given in separate base systems, i.e. parameters beam waist size and beam waist position associated with the two beams differ. Thus, on interference and probably for the resulting beam no optimum base system can be defined. Consequently the phase information of the beam is spread of a number of transversal modes. Mathematically this does not pose a problem, as long as a sufficiently large number of higher modes are used to describe the beam. However, in practice the definition of operating points becomes much more difficult. And much more care is required to assure the simulation is set up correctly. The following sections illustrate the problem with some examples and give some advice on how to use FINESSE in the presence of mode mismatching.

4.10.1 Phases and operating points

On operating point can be defined as the microscopic positions of the interferometer optics. More precisely it is given by the phases of light fields at the location (optical surface) of interference.

In FINESSE (and many other numeric simulations) the parameters accessible by the user include the microscopic positions of optical surfaces but not the phases of light fields. The latter describe an output of the simulation rather than an input. Thus it is up to the user to define the microscopic position of optical surfaces such that the light fields feature the correct phase upon interference.

FINESSE tries to ease this task by several measures, some of which are optional. The following features reflect design choices which apply to both modes (plane wave and Hermite-Gauss):

- The length of space components is defined as a inter multiple of the default wavelength λ. Without Hermite-Gauss modes, i.e. when the Gouy phase is not considered, this ensures that a space is 'resonant' to the carrier field, i.e the phase of the field leaving the space is the same as on entering it.
- Microscopic positions are given as *tunings* which provides an intuitive user input as often operating points can be set with tunings of 0, 45 or 90 degrees.

In the Hermite-Gauss mode the following two simplifications can be used:

- The `cav` command and the automatic beam trace routine allows to use cavity eigenmodes wherever possible.
- The `phase` command can be used to zero the Gouy of the TEM_{00} mode and of the coupling coefficients for the TEM_{00} mode, see below.

And most importantly the `lock` command provides the means to reach the operating point accurately when the operating point cannot easily be set by the user manually.

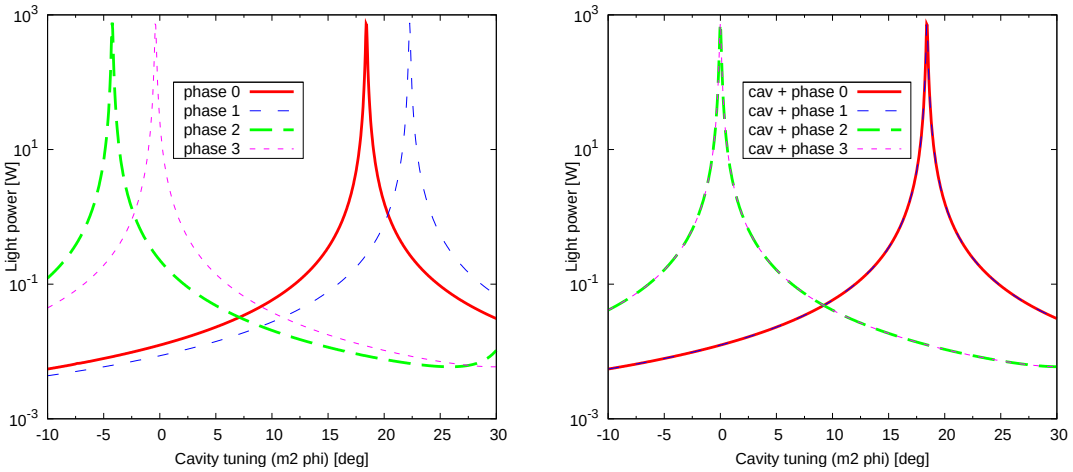


Figure 4.6: *The light power in the cavity while one mirror in tuned around the cavity resonance. The left plot was created not using the* `cav` *command whereas the right plot shows the results obtained using* `cav` *and thus the cavity's eigenmodes. The different simplifications set by the* `phase` *command change the mirror tuning at which resonance is achieved. The respective offset values are listed in Table 4.1.*

4.10.2 The `phase` command and its effects

The `phase` command can be used to switch on/off some simplification with respect to the phase of the light field. The syntax is as follows:

- phase 0: No simplification. This means for example that the Gouy phase of a TEM_{00} is *not* zero and thus a space of arbitrary length is in general not resonant to the carrier field
- phase 1: The phase of coupling coefficients is shifted so that the phase of k_{00} is 0. The phases of all coupling coefficients k_{nm} for one field coupling, for example, a reflection at one side of a mirror, are changed by the same amount. To some effect that resembles the movement of the optical surface. However, since this is independently applied to all coupling coefficients of the surface (e.g. two reflections and two transmissions), this does not describe a possible real situation. In fact, it might validate energy conservation or produce other weird effects.
- phase 2: The phase accumulated in a 'space' components is adjusted to that the phase of TEM_{00} set to 0. This simply removed the effect from the Gouy phase on the 'resonance' of the space components, i.e. the length of a space is made to be *not* anymore a integer multiple of the wavelength in order to produce the desired effect of 'resonance' for the TEM_{00} mode.
- phase 3: Both of the above (1+2)

Currently the default setting in FINESSE is `phase 3`. The motivation for this has been to provide the beginner with default settings that yield intuitive results straight away. Experienced users should check whether they can develop the habit of using `phase 2`

instead which can be a bit laborious to use but always produces physically correct results.

phase	0	1	2	3	0	1	2	3
cav	no	no	no	no	yes	yes	yes	yes
offset	18.4	22.3	-4.2	-0.4	18.4	18.4	0.0	0.0

Table 4.1: *Tuning offsets for different use of the* `cav` *and* `phase` *command. The numbers shown here correspond to the graphs in Figure 4.6.*

The effects of the `phase` command on the operating point of a simple Fabry-Perot cavity is shown in Figure 4.6. In this and the following examples the cavity parameters are set to:

cavity length	Rc m1	Rc m2	T m1	L m1	T m2	L m2
3995 m	-2076 m	2076 m	$5e^{-3}$	$4.5e^{-6}$	$10e^{-6}$	$50e^{-6}$

With m1 being the input mirror and m2 the end mirror of the cavity. These parameters represent a cavity similar to a LIGO/AdLIGO arm cavity.

Figure 4.6 shows that the operating point is strongly dependent on the use of the `phase` and `cav` commands. The offsets represented as tunings of the end mirror are listed in Table 4.1. The numbers present the tuning which need to be set by the user to set the cavity on resoance (assuming a tuning of 0 degrees for the input mirror).

Figure 4.7 demonstrates another curious effect of the phase command. In the case of a simple two mirror cavity one can show that `phase 3` and `phase 1` can violate energy conservation. In this example the input beam is not mode-matched to the cavity, the calculation is performed using cavity eigenmodes and the operating point has been adjusted manually. Even though the effect is small and would probably not be noticeable in many simulation results, it shows that the `phase` command must be used with care.

In summary, a simple intuitive number to be set manually by the user can only be achieved when using eigenmodes and `phase 2/3` (it should be noted that in the presence of a mode mismatch at a beam splitter only `phase 3` will provide an intuitive tuning for the operating point). However, physically correct results are only guaranteed with `phase 2/0`.

4.10.3 Mode mismatch effects on the cavity phase

In high-finesse cavities a small change of the light phase can quickly detune the cavity. For example, a mathematical mode-mismatch *inside* the cavity, which occurs when the beam parameters used in the calculation are not exactly equal those of the circulating beam, can easily lead to wrong results and has to be treated with care.

Figure 4.8 shows the power inside a linear cavity as a function of the radii of curvature of the cavity mirrors (the cavity is symmetric). The importance of using cavity eigenmodes is demonstrated by the fact that the correct results (in this example) are only achieved by either using many higher-order modes, preferably with a `lock` command, or by using

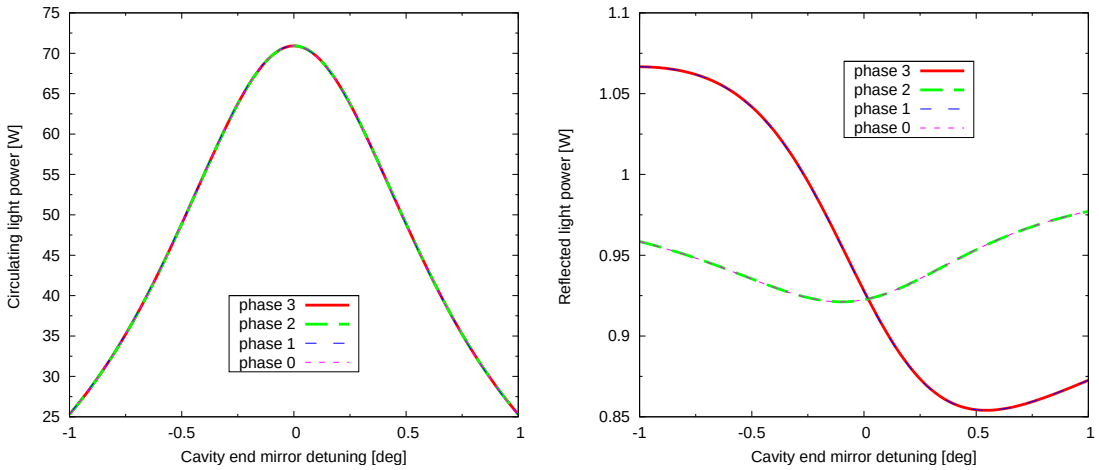


Figure 4.7: *The two plots show the circulating power (left) and the reflected power (right) of a Fabry-Perot cavity for different* `phase` *settings. The input power is set to 1 W. The two phase settings which reset the phase of the coupling coefficients (* `phase 3` *and* `phase 1`*) can 'create' energy as the reflected light can be larger than the injected light. This illustrates that they can produce non-physical results whereas* `phase 2` *and* `phase 0` *do not show this problem. In these examples the cavity is set to use eigenmodes and the operating point has been adjusted manually. The input beam is not matched to the cavity modes and* `maxtem 2` *has been used. Please note, the parameters of Table 4.1 were* not *used to generate these plots, arbitrary parameters have been chosen instead to demonstrate the effect.*

the cavity eigenmodes. Note that in this case the results do not depend on the setting of the `phase` command.

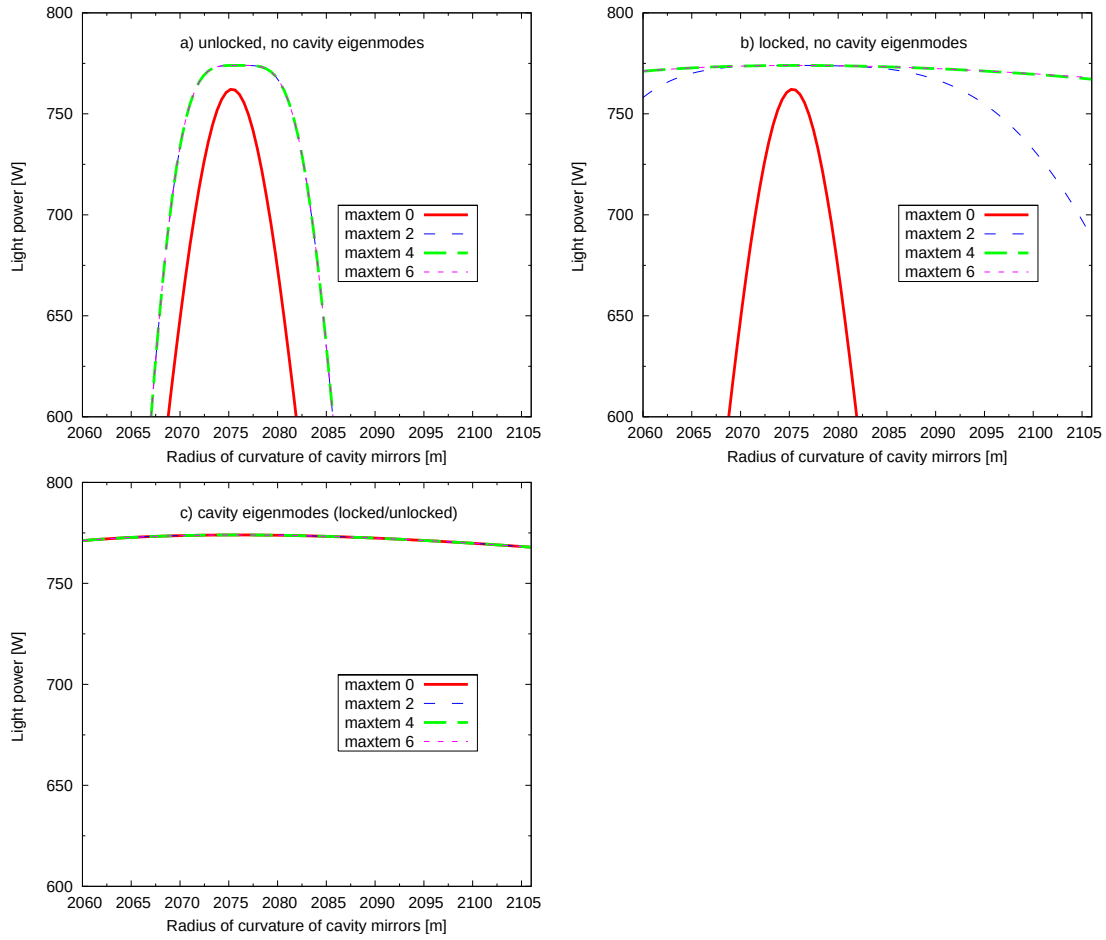


Figure 4.8: *The three graphs compare the cavity power as a function of the radii of curvature of the cavity mirrors (the radii of curvature are changes symmetrically, i.e. $Rc_1 = -Rc_2$). The results do not change for different usage of the* `phase` *command. The input beam has been set to be matched to the eigenmodes of the cavity for $Rc = 2076\,\mathrm{m}$. Graph a) shows the situation when the* `cav` *command is not used and thus the system in analysed with a non-opimal base. A very rapid drop of the cavity power can be observed when the radius of curvature deviates from 2076 m. This is due to a change in the operating point of the cavity. This can be easily shown by redoing the simulation using a* `lock` *command to keep the cavity on resonance, as shown in graph b). In this case, using the cavity eigenmodes produces always the correct result without the need for using a lock or higher order modes, as shown in graph c). Of course a mode mismatch at a beam splitter does not have a proper base system like a single cavity. In that case a lock is often the only way to ensure correct results.*

4.11 Misalignment angles at a beam splitter

The coupling of Hermite-Gauss modes in a misaligned setup as described above is defined by a misalignment angle. However, in the case of a beam splitter under arbitrary incidence the analysis of the geometry is complicated because it is commonly described in three different coordinate systems. The purpose of this section is to derive a precise description of the problem. FINESSE uses an approximation and the calculations below can be used to estimate the (very small) error of that approach.

Our discussion will be limited to the following setup: a beam travelling along the z-axis (towards positive numbers) and a beam splitter (surface) located at $z = 0$, which may be rotated around the y-axis by an angle α ($|\alpha|$ = angle of incidence). This shall be the 'aligned' setup.

To describe a misalignment of the beam splitter, one usually refers to a coordinate system attached to the beam splitter. This coordinate system is called (x', y', z') in the following and can be derived—in this case—by rotating the initial coordinate system by α around the y-axis. The misalignment can be quantified by two angles β_x, β_y that describe the rotation of the beam splitter around the x'-axis and the y'-axis, respectively. Rotation around the x'-axis is often called *tilt*, and rotation around the y'-axis simply *rotation*. Whereas the initial rotation α may be large, the misalignment angles β_x and β_y are usually small. In fact, most models describing the effects of misalignment use approximations for small perturbations.

Here we are interested in the exact direction of the reflected beam. The reflected beam, though, may be characterised in yet another coordinate system (x'', y'', z'') with the z''-axis being parallel to the reflected beam. This coordinate system can be derived from (x, y, z) by a rotation of 2α around the y-axis. A misalignment of the beam splitter will cause the beam to also be misaligned. The misalignment of the beam is given by the two angles γ_x, γ_y that describe the rotation around the x''-axis and the y''-axis, respectively.

It can easily be shown that for $\beta_x = 0$, the misalignment of the beam is $\gamma_x = 0$ and $\gamma_y = 2\beta_y$. For normal incidence ($\alpha = 0$) we get a similar result for $\beta_y = 0$: $\gamma_y = 0$ and $\gamma_x = 2\beta_x$. For arbitrary incidence, the geometry is more complex. In order to compute the effect caused by a tilt of the beam splitter we need basic vector algebra. Please note that the following vectors are given in the initial coordinate system (x, y, z). First, we have to compute the unit vector of the beam splitter surface $\vec{e}_{\mathrm{bs}}$. This vector is rooted at (0,0,0), perpendicular to the surface of the beam splitter and pointing towards the negative z-axis for $\alpha = 0$.

For $\alpha = 0$ this vector is $\vec{e}_{\mathrm{bs}} = -\vec{e}_z$. Turning the beam splitter around the y-axis gives:

$$\vec{e}_{\mathrm{bs}} = (\sin(\alpha), 0, -\cos(\alpha))\,. \tag{4.119}$$

Next, the beam splitter is tilted by the angle β_x around the x'-axis. Thus, the surface vector becomes:

$$\vec{e}_{\mathrm{bs}} = (\sin(\alpha)\cos\beta_x, -\sin(\beta_x), -\cos(\alpha)\cos\beta_x)\,. \tag{4.120}$$

In order to calculate the unit vector parallel to the reflected beam, we have to 'mirror' the unit vector parallel to the incoming beam $-\vec{e}_z$ at the unit vector perpendicular to the beam splitter. As an intermediate step, we compute the projection of $-\vec{e}_z$ onto $\vec{e}_{\text{bs}}$ (see Figure 4.9):

$$\vec{a} \;=\; -(\vec{e}_z \cdot \vec{e}_{\text{bs}})\;\vec{e}_{\text{bs}} = \cos(\beta_x)\cos(\alpha)\vec{e}_{\text{bs}}. \tag{4.121}$$

The reflected beam ($\vec{e}_{\text{out}}$) is then computed as:

$$\begin{aligned} \vec{e}_{\text{out}} &= -\vec{e}_z + 2(\vec{a} + \vec{e}_z) = 2\vec{a} + \vec{e}_z \\ &= (2\cos^2(\beta_x)\cos(\alpha)\sin(\alpha), -2\cos(\beta_x)\cos(\alpha)\sin(\beta_x), -2\cos^2(\beta_x)\cos^2(\alpha) + 1) \\ &=: (x_o, y_o, z_o). \end{aligned} \tag{4.122}$$

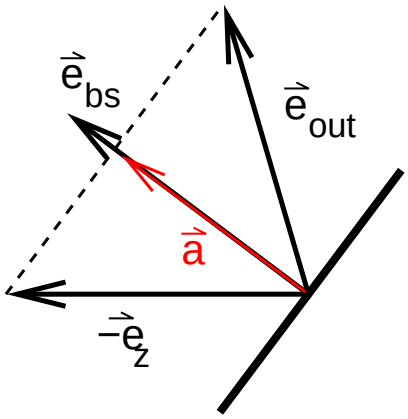


Figure 4.9: *Mirroring of vector $-\vec{e}_z$ at the unit vector of the beam splitter surface $\vec{e}_{bs}$.*

To evaluate the change of direction of the outgoing beam caused by the tilt of the beam splitter β_x, we have to compare the general output vector $\vec{e}_{\text{out}}$ with the output vector for no tilt $\vec{e}_{\text{out}}\big|_{\beta_x=0}$. Indeed, we want to know two angles: the angle between the two vectors *in* the x-z plane (γ_y), and the angle between $\vec{e}_{\text{out}}$ and the x-z plane (γ_x). The latter is simply:

$$\sin(\gamma_x) = 2\cos(\beta_x)\cos(\alpha)\sin(\beta_x) = \cos(\alpha)\sin(2\beta_x). \tag{4.123}$$

For small misalignment angles ($\sin(\beta_x) \approx \beta_x$ and $\sin(\gamma_x) \approx \gamma_x$), Equation 4.123 can be simplified to:

$$\gamma_x \approx 2\beta_x \cos(\alpha). \tag{4.124}$$

One can see that the beam is tilted less for an arbitrary angle of incidence than at normal incidence. An angle of 45°, which is quite common, yields $\gamma_x = \sqrt{2}\beta_x$.

In order to calculate γ_y, we have to evaluate the following scalar product:

$$\begin{aligned} &\vec{e}_{\text{out}}\big|_{y_o=0} \cdot \vec{e}_{\text{out}}\big|_{\beta_x=0} = \sqrt{x_o^2 + z_o^2}\cos(\gamma_y), \\ &\Rightarrow \qquad \cos(\gamma_y) = \tfrac{-1}{\sqrt{x_o^2+z_o^2}}\left(x_o\sin(2\alpha) + z_o\cos(2\alpha)\right). \end{aligned} \tag{4.125}$$

This shows that a pure tilt of the beam splitter also induces a rotation of the beam. The amount is very small and proportional to β_x^2. For example, with $\alpha = 45°$ and $\beta_x = 1\,\text{mrad}$, the rotation of the beam is $\gamma_y = 60\ \mu\text{rad}$. Figure 4.10 shows the angles γ_x and γ_y as functions of α for $\beta_x = 1°$.

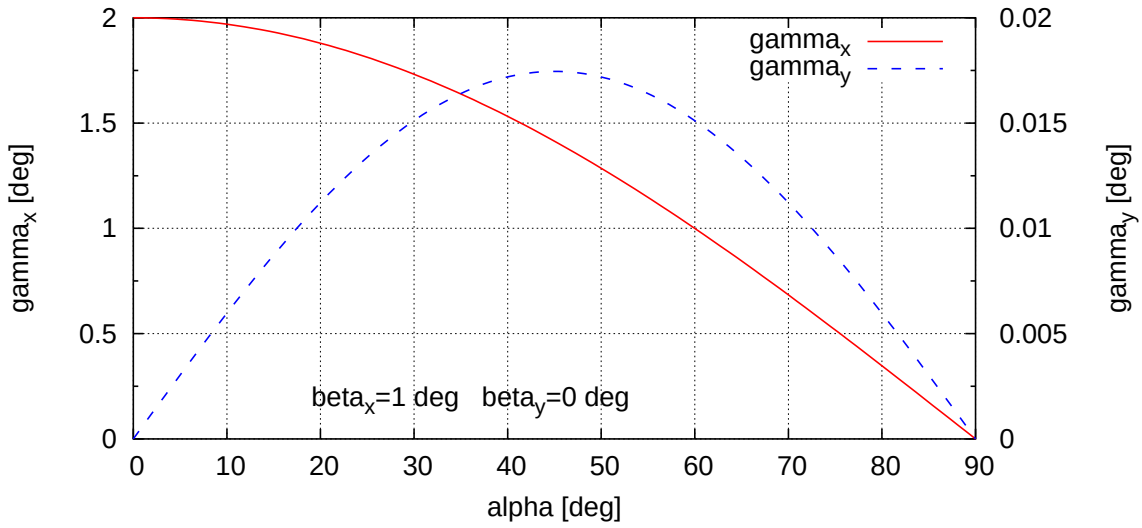


Figure 4.10: *Misalignment angles of a beam reflected by a beam splitter as functions of the angle of incidence α. The beam splitter is misaligned by $\beta_x = 1°$ and $\beta_y = 0$. Note that the values for β are very large, see Section 4.9, and have been chosen to artificially enlarge the coupling between vertical and horizontal misalignment for this demonstration.*

In the case of $\beta_x \neq 0$ and $\beta_y \neq 0$, the above analysis can be used by changing α to $\alpha' = \alpha + \beta_y$.

In FINESSE the coupling between β_x and β_y is ignored. In other words, the effect shown by the red (solid) trace in Figure 4.10 is included in FINESSE whereas the effect illustrated by the blue (dashed) trace is not.

4.12 Aperture effects and diffraction losses

By default FINESSE assumes that all optical components have an infinite size transverse to the optical axis, however you are can specify the radius of mirrors using the command:

```
attr mirror_name r_ap value
```

where `value` is the aperture radius in metres. It is also possible to vary the aperture radius with the `xaxis` command by specifying a mirror and using the `r_ap` attribute. The effect of an aperture is calculated using higher order modes, therefore when an aperture is defined a coupling coefficient matrix is computed. This requires using the

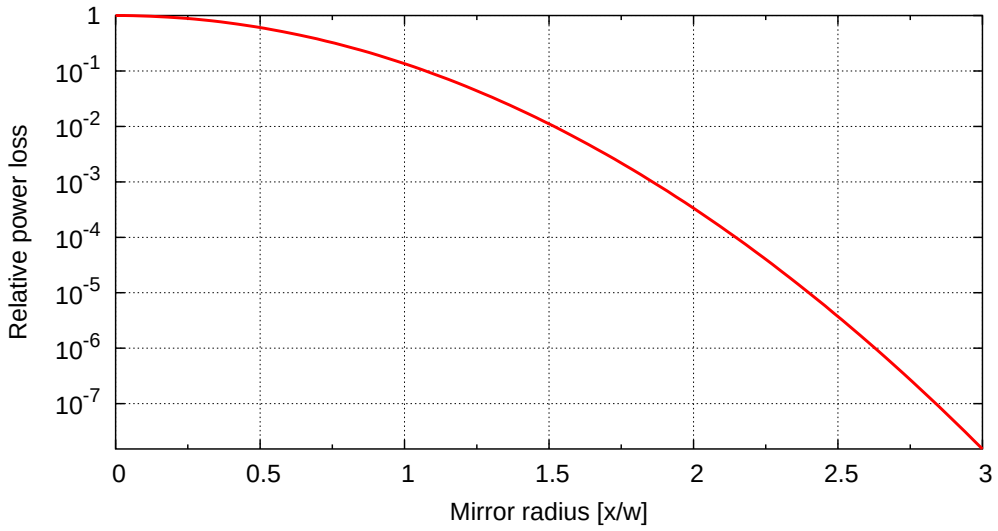


Figure 4.11: *The plot shows a lower limit of the power loss experienced by a beam of size w on reflection at a mirror of diameter $2x$.*

`maxtem` command to choose the maximum mode order to use in the calculations. Below we see an example on how choosing `maxtem` value affects the power loss.

Note that currently it is only possible to calculate the coupling coefficients by computing the integral from equation 4.78 numerically.

From the intensity profile given in Equation 4.4 we can compute the amount of power with respect to a given area. The power inside a circular disk with the radius x (with the centre on the optical axis) can be computed as:

$$\begin{aligned} P_{\text{disk}} &= \int_{\text{disk}} I(r) = \int_0^{2\pi} d\phi \int_0^x dr\, r I(r) \\ &= \tfrac{4P}{w} \int_0^x dr\, r e^{-2r^2/w^2} \\ &= -P \int_0^x dr \partial_r e^{-2r^2/w^2} = -P \left[e^{-2r^2/w^2} \right]_0^x \\ &= P \left(1 - e^{-2x^2/w^2} \right). \end{aligned} \tag{4.126}$$

A beam that is reflected at a mirror with diameter $2x$ will thus experience a power loss of at least:

$$P_{\text{loss}} = P\, e^{-2x^2/w^2}. \tag{4.127}$$

This is almost the same distribution as the intensity itself. With respect to losses we are interested in small deviations from P and look at the distribution in a different way. Figure 4.11 shows the amount of power lost as a function of x/w (the mirror *radius* with respect to the beam radius). In modern high finesse cavities where losses due to surface and coating imperfections can be in the range of a few ppm, the mirror's diameter should

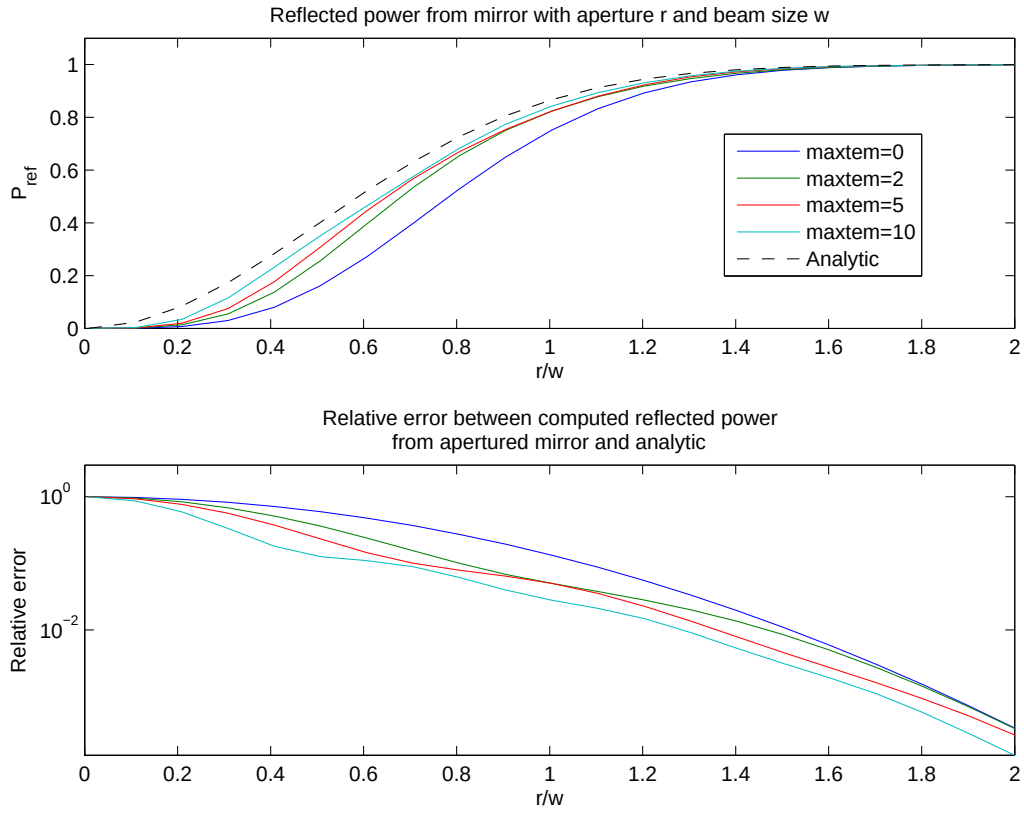


Figure 4.12: *Shown in the top plot is the power reflected from an apertured mirror with different maxtem options. Analytic refers to equation 4.126. The bottom plot shows the difference between the maxtem results and the analytic result.*

at least be 2.5 times the beam diameter. Typically mirrors are designed to be at least three times the nominal beam diameter including a safety margin allowing for imperfect alignment and some changes in the beam diameter.

This calculation is meant for deriving limits only. In general, the effects of apertures have to be analysed taking into account the effects of diffraction and higher order modes.

Modelling a sharp aperture, such as a finite sized mirror, with modes or any paraxial method, is not ideal. For example, representing a perfectly sharp cut-off would require an infinite number of modes. It is therefore advisable to keep in mind the amount of power that is lost due to the aperture and also the fact a finite number of modes are used. Below a simple test was run that looked at the power reflected from a mirror, we then vary the size of the aperture and plot the reflected power for increasing maxtem.

```
l i1 1 0 n1
s s1 1 n1 n2
m m1 1 0 0 n2 dump

pd0 Pref n2
```

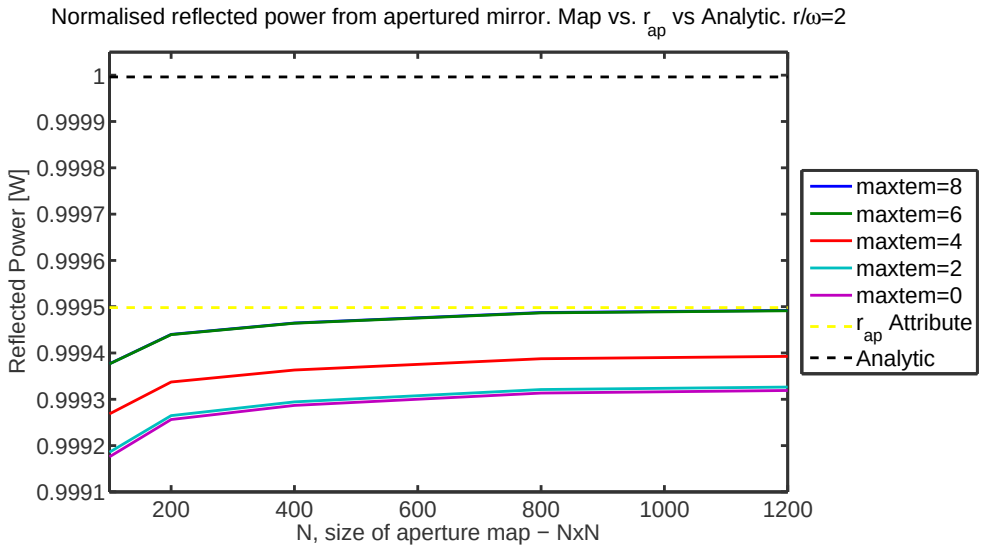


Figure 4.13: *Here the performance of an aperture mirror map is tested against the analytic solution and using the r_{ap} attribute of the mirror. The r_{ap} attribute was computed using maxtem 8. Here we can see a high resolution map is required to reach the same levels of accuracy. Both do not reach the analytic value though.*

```
gauss g1 m1 n2 1e-3 0
conf m1 knm_flags 8 # need this to force aperture integration
conf m1 integration_method 3 # use cuba parallel integrator
xaxis m1 Rap lin 1e-5 2e-3 40
# can also be applied using attr for a single value
# attr m1 Rap 1e-4
maxtem 5
```

The results are plotted in figure 4.12 which demonstrates that even using maxtem 20 is not sufficient to fully represent the beam. Using such a high value for maxtem causes the simulation computation to become very slow and it is not generally recommended; a maxtem of 10 appears to offer a good level of accuracy and is still relatively quick to compute in comparison. We can estimate the amount of extra power we are missing compared to what the analytic equation predicts by using the bottom plot in figure 4.12.

It should also be noted that aperture effects can also be computed by using absorption mirror maps, which have a resolution $N \times N$ with width and height of the map of $2r_{ap}$ (Thus a circular aperture fits perfectly within the map). However as shown in Figures 4.13 and 4.14 the map must be of a high enough resolution to provide accurate results at ppm levels, due to circlular aperture not being perfectly represented in a rectangular grid. Using large maps also has computational performance issues whereas using the r_{ap} attribute can be comparitviely quick to compute. Both r_{ap} and the maps are computed using the same routines, the former however does so in a more efficient manner, thus we recommend when possible using the r_{ap} attribute.

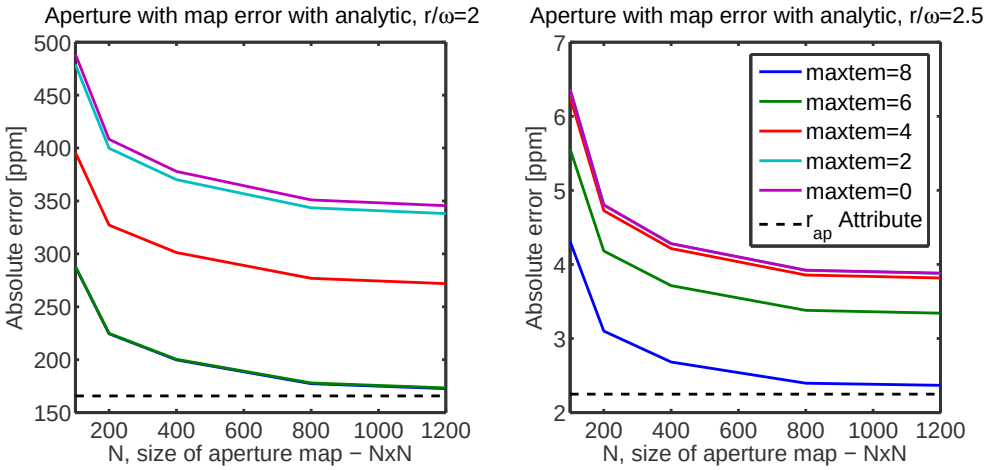


Figure 4.14: *The level of accuracy possible is limited by the ratio between the aperture size r and the beam size ω. Unless a high resolution aperture map is used results differ significantly. r_{ap} line is using maxtem 8.*

Chapter 5

Radiation pressure

Interferometric gravitational wave detectors make use of suspension systems to decouple mirrors from ground motion. Furthermore these interferometers operare with a very high circulating light power so that the radiation pressure from the reflected photons can significantly influence the mechanical behaviour of the mirrors. This leads to a very interesting coupling between the optical fields and the mirrors as mechanical oscillators, forming so-called *optical springs*. Optical springs and other radiation pressure effects can change the dyanmics of an interferometer significantly, especially when considering transfer functions from mechanical motions to optical outputs. The implementation of radiation pressure effects in FINESSE has been a long requested feature and is now available with the release of FINESSE 2.0.

5.0.1 Radiation pressure calculation approximations

It is important, as with any simulation tool, to understand the limits of the approximations made to allow radiation pressure effects to be implemented in FINESSE. By nature radiation pressure is proportional to the beam power, a non-linear relation in regards to the amplitude of the optical fields which is what FINESSE computes. To linearise the the physical equations described in the following sections the following assumptions are made::

- The magnitude of any motion in the mirrors surface is much smaller than the wavelength of light, by default, 1064nm.
- Therefore the optical sidebands created by such motion are also much smaller in magnitude than the magnitude of any carrier (laser or RF modulation) field.
- That the frequency difference between carrier fields, laser fields and RF sidebands created with `modulator` components, are large in comparison to the frequencies of signals and of mechanical resonances of optics. This is required so that any beating between carriers fields does not cause significant radiation pressure effects.

It also follows that the maximum signal frequency must be less than half of the minimum frequency difference between carrier fields. If this frequency limit is exceeded, it is possible for the upper sidebands of one carrier field to become the lower sideband of another. This is not a scenario that the radiation pressure implementation of FINESSE is designed to handle. Given that `modulator` frequencies are typically the order of MHz and radiation

pressure effects are only of interest at low frequencies due to the mechanical susceptibilty of a suspended object being $\propto 1/f^2$, this should not be an issue that comes up frequently.

5.1 Radiation pressure force to a mirror motion

Any electromagnetic field will exert a force on a suspended optic, the frequency spectrum of such a force from one beam is given as,

$$F(f) = \frac{P(f)\cos(\alpha)}{c}. \tag{5.1}$$

Where $P(f)$ is the fluctuation of the power in the laser at frequency f, α is the angle of incidence and c the speed of light. FINESSE models only consider forces with $f > 0$, in other words we assume that the DC radiation pressure force to be compensated for by some control system that keep the mirror in the same position.

Any fluctuation in the light power at a given frequency f_s (signal frequency) can be described by a positive and negative part in the frequency domain,

$$P(f_s) \quad = \quad P_s^+ e^{i2\pi f_s t} + P_s^- e^{-i2\pi f_s t}. \tag{5.2}$$

As $P_s^+ = P_s^{-*}$ there is no need to explicitly solve P_s^- in any of our simulations. Instead we only compute the positive frequency components. Assuming that all carrier fields possess both an upper $a^+_{s,jnm}$ and lower $a^-_{s,jnm}$ sideband of mode TEM_{nm} and which belongs to the j^{th} carrier field, the fluctuating power is then given by,

$$P_s^+ \quad = \quad \sum_j \sum_{n,m} \left(a^+_{s,jnm} a^*_{c,jnm} + a^{-*}_{s,jnm} a_{c,jnm} \right). \tag{5.3}$$

It should be noted that this power fluctuation calculation depends on both the upper sideband and the conjugate of the lower sideband – conjugation is not a linear operation so cannot be represented in a matrix form of the equations. By choosing to model only the positive frequencies we must now propagate the conjugate of the lower signal sidebands in the interferometer matrices that we construct. Although internally this requires constructing the matrices differently, any output from FINESSE will always output the non-conjugated lower sideband, such as if you use the `ad` detectors.

The total force applied to a suspended mirror is then the sum of all forces due to each incoming and outgoing beams present at the optical component; for a mirror this is four beams and at a beamsplitter it is eight,

$$F_{total}(f) \quad = \quad -F_{1i}(f) - F_{1o}(f) + F_{2i}(f) + F_{2o}(f), \tag{5.4}$$

$$\quad = \quad \cos(\alpha)\frac{-P_{1i}(f) - P_{1o}(f) + P_{2i}(f) + P_{2o}(f)}{c}. \tag{5.5}$$

The $\pm$ for each P_s is determined by which side of the mirror or beamsplitter the beam is on: The positive direction of motion, the surface normal, for the mirror is defined as the direction of the beam reflected by the side of the first node. Thus an incoming or

outgoing beam imparts a negative momentum on the side of the first node and a positive on the side of the last.

As we only compute the output for a single value of f_s in each step we set $F_s \equiv F_{total}(f)$ and $P_x(f) \equiv P_{s,x}$ leaving,

$$F_s = \frac{\cos(\alpha)}{c}\left(-P_{s,1i} - P_{s,1o} + P_{s,2i} + P_{s,2o}\right). \tag{5.6}$$

Variables subscripted with an s are typically frequency dependent terms which are being computed at an assumed signal frequency f_s. The longitudinal motion at the signal frequency is then found using,

$$Z_s = H_s \sum_{n}^{N_F} F_{s,n}. \tag{5.7}$$

Where H_s is the transfer function (See section 3.1.2) for the mechanical response of the susepended optic due to a force being applied to it in the z direction. This sum of forces can also contain other external forces being applied, such as from actuators.

5.2 Mirror motion to optical phase change

Evaluating equation 5.7 determines the amplitude and phase of the mirror oscillations. When carrier fields and RF modulation sidebands are reflected from a moving mirror this oscillation will produce phase modulation signal sidebands, $a^{\pm}$, around these fields. The variation in height of a mirror's surface at a frequency f_s is described by $z_s(x,y)$, this function is a normalised description of the general motion; its amplitude is given by A_s, where $A_s^+ \equiv A_s$ and $A_s^- \equiv A_s^*$ (as lower sideband computation requires the conjugate of the motion amplitude). For example, longitudinal motion is described by $z_s(x,y) = 1$ and thus $A_s = Z_s$ is in meters. Rotational motion is defined by $z_s(x,y) = x$ or y with $A_s = \Theta_{x/y}$ in radians. Making the assumption that $|A_s| \ll \lambda$ 1064nm, the creation of signal sidebands is linearised,

$$a^{\pm}_{s,jnm} = \frac{irkA_s^{\pm}}{\cos(\alpha)} \sum_{n',m'} a_{c,jn'm'} \iint_{\infty}^{\infty} u_{n'm'}(x,y) e^{i2kz_o(x,y)} z_s(x,y) u^*_{nm}(x,y) dxdy \tag{5.8}$$

We find that the creation of sidebands can be described using coupling matrices, where K^o is the static surface distortion present and K^s is the coupling caused by the motion of the surface,

$$K^o_{nmn'm'} = \iint_{\infty}^{\infty} u_{n'm'}(x,y) e^{i2kz_o(x,y)} u^*_{nm}(x,y) dxdy, \tag{5.9}$$

$$K^s_{nmn'm'} = \iint_{\infty}^{\infty} u_{n'm'}(x,y) z_s(x,y) u^*_{nm}(x,y) dxdy, \tag{5.10}$$

$$a^{\pm}_{s,jnm} = \frac{irkA_s^{\pm}}{\cos(\alpha)} \sum_{n',m'} a_{c,jn'm'} (K^s K^o)_{nmn'm'} \tag{5.11}$$

Longitudinal motions are relatively simple to compute as the $K^s = I$, meaning that the proportional content of the incoming carrier fields is replicated in the phase modulation signal sidebands.

$$z_s(x,y) = 1 \tag{5.12}$$

$$K^s_{nmn'm'} = \delta_{nn'}\delta_{mm'}, \tag{5.13}$$

$$a^{\pm}_{s,jnm} = \frac{irk}{\cos(\alpha)} Z^{\pm}_s \sum_{n',m'} a_{c,jn'm'} K^o_{nmn'm'} \tag{5.14}$$

Further higher-order motions like rotations require an analytic solution to K^s which can be found in [Brown].

5.3 Example: optical spring

In this examine we show how to suspend a cavity and to readout the transfer function from a force (applied to the suspended mirrors) to the mirrors motion.

Radiation pressure inside a suspended cavity can create so-called optical springs, which are created from combination of the radiation pressure force and the usual restoring force due to gravity. An optical spring is created when the cavity is detuned so that the power in the cavity depends linearly on the change in the mirror position. Optical springs can be stable or unstable, depending on the direction of the detuning. Optical springs can cause interesting effects in suspended interferometers and with Finesse we can model how the motion mirror motions and optical signals are affected by them.

Firstly we define a simple optical cavity using just plane waves for now. We have detuned the far mirror of the cavity, to allow for an optical spring:

```
l l1 3 0 n1a
s s1 0.5 n1a n1b
m Min 0.9937 0.0063 0 n1b n2
s cav1 1 n2 n3
m Mend 1 0 -0.048 n3 n4
```

To make the cavity mirrors suspended all we need to do is specify a mass for the mirrors using the `attr` command:

```
attr Min  mass 0.25 # kg
attr Mend mass 0.25 # kg
```

By default any mirror or beamsplitter given a mass becomes a *free mass*, i.e. the mechanical response to a force applied to it in the axis of the beam is $\propto 1/f^2$, where f is the frequency of the force applied. Simple but realistic or more complex suspensions systems can also be modelled by describing the mechanical response – or transfer function from force to motion – via poles, zeros and quality factors. For this example we define a transfer function for a suspension system that has a resonance at 1 Hz with a quality factor

of $Q = 10^5$. Multiple suspension setups can be create and are applied to a suspended mirror or beamsplitter using the `attr` command:

```
tf sus 1 0 p 1 100000

attr Min zmech sus
attr Mend zmech sus
```

Now that we have our suspended cavity setup we want to output the motions of these mirrors. This is done using the motion detector, `xd`. Here we specify which suspended mirrors we want to output and which of its motions, the z motion in this case::

```
xd ETM Mend z
xd ITM Min z
```

A motion of a suspended mirror can be induced in two ways: by applying a mechanical force with `fsig`, or via some modulated optical field, e.g. an amplitude modulated carrier field. For this example we will apply a force to the far mirror of the cavity, setting the target property of the `fsig` to `Fz`, i.e. a force to the z motion.

```
fsig aforce Mend Fz 1 0 1

xaxis aforce f log 0.1 1k 1000
yaxis log abs:deg
```

We scan the frequency of the applied force and plot the magnitude and phase of the mirrors' motion, the result is shown in figure 5.1. We can see that even though we are only applying a force the far mirror the input mirror is also moving due to the optical spring effect, i.e. the strong circulating light field coupling the two mirrors mechanically. If the cavity is put exactly on resonance then the input mirror will not move. There are two peaks in the motions, the one at 1 Hz is the suspension resonance and the second at 80 Hz is the optical spring resonance. From the phase of the motion we can also tell that at frequencies below 50 Hz the mirrors move in a common mode with equal magnitude, this is commonly referred to as optical rigidity. Above this frequency the mirrors move differentially or out-of-phase.

The complete FINESSE file for this example is then:

```
l l1 3 0 n1a
s s1 0.5 n1a n1b
m Min 0.9937 0.0063 0 n1b n2
s cav1 1 n2 n3
m Mend 1 0 -0.048 n3 n4

tf sus 1 0 p 1 100000

attr Min m 0.25 zmech sus
attr Mend m 0.25 zmech sus
```

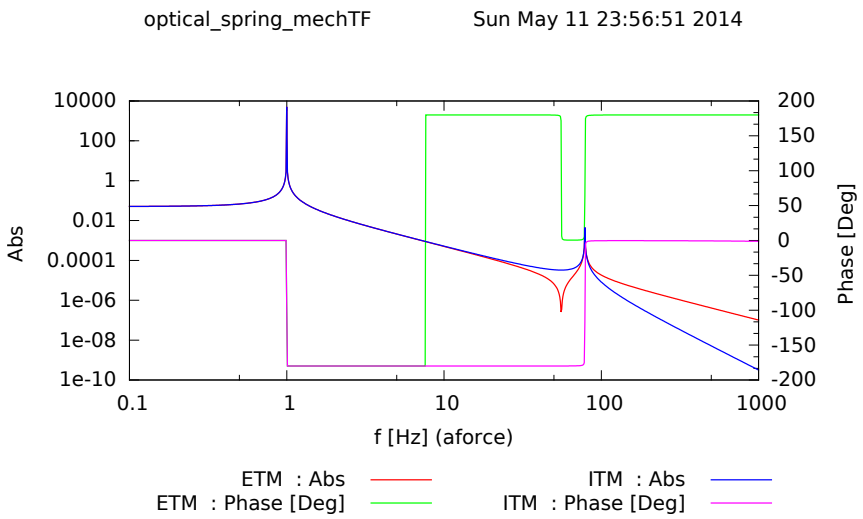


Figure 5.1: *Transfer function from force applied to cavity end mirror to motion of both cavity mirrors.*

```
fsig aforce Mend Fz 1 0 1

xd ETM Mend z
xd ITM Min z

xaxis aforce f log 0.1 1k 1000
yaxis log abs:deg
```

5.4 Rotational mirror motion

Rotational mirror motion is essentially the next higher-order motion possible after longitudinal motion along the beam axis. The rotations implemented in FINESSE are yaw and pitch – which are the dynamic equivalent to the `xbeta` and `ybeta` DC rotations of a mirror or beamsplitter respectively. Again, a mirror or beamsplitter becomes suspended in rotation when we state the optic's moment of inertia with either the I_x for yaw or I_y for pitch. As with longitudinal motion, more complex suspensions can also be specified with transfer functions from torques to rotations. A simple example:

```
attr component_name Ix 0.25
attr component_name Iy 0.25
attr component_name rxmech transfer_function
attr component_name rymech transfer_function
```

5.5 Example: torsional optical spring

Below we will plot the transfer function from a torque applied to the end mirror of a cavity to both of the mirrors' yaw motions. For this we specify the moment of inertia for each mirror, the transfer function for the suspension and use the `xd` detectors to read out the yaw motions. The torque is applied to the mirror using the `fsig` command targetted at the mirror's `Frx` property, i.e. force in the rotational x-z plane motion.

```
tf rxpend 1 0 p 0.6 1000

l l1 100 0 n1
m Min 0.99 0.01 0 n1 n2
s cav 4000 n2 n3
m Mend 1 0 0 n3 n4

attr Min Ix 1 rxmech rxpend
attr Mend Ix 1 rxmech rxpend

attr Min Rc -2076
attr Mend Rc 2076

cav c1 Min n2 Mend n3

fsig aforce Mend Frx 1 0 1

xd rx_in Min rx
xd rx_end Mend rx

xaxis aforce f log 2e-1 3 1000
yaxis log abs:deg
maxtem 1
```

The output is shown in figure 5.2. In this we can see two resonance peaks, these are due to the soft and hard rotational modes of a cavity: soft is when the mirrors rotate out-of-phase with one another, which is an unstable motion; the hard mode is when the rotational motion is in-phase. The suspension resonance at 0.6 Hz is no longer at exactly that frequency due to it being shifted by the opto-mechanical coupling.

5.6 General of surface motions

Longitudinal and rotational motions are the most commonly used and the analytically simple nature of $z_s(x, y)$ means that analytic solutions for K^s are available. For more complicated motions, such as surface motions as a result from internal modes of the mirror's substrate being excited, the individual elements of K^s must be computed via a numerical integration routine.

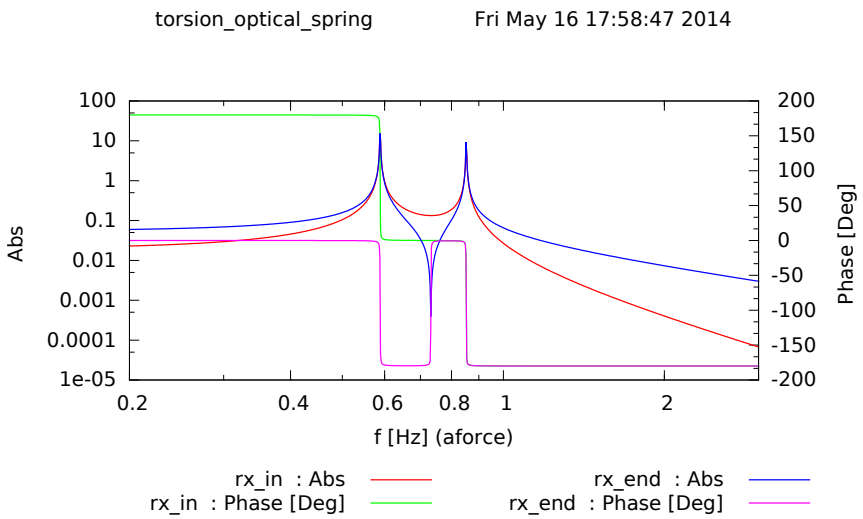


Figure 5.2: *Transfer function from torque applied to cavity end mirror to yaw motion of both cavity mirrors. Plot shows both soft (first resonance) and hard (second resonance) mode motions.*

As there is no direct analog of mass or moment of inertia for a general motion, you need only specify the motion for it to become "suspended", for this we use the `smotion` (surface-motion) command (currently only available for mirror components, not beamsplitters):

```
smotion component map_filename transfer_function
```

The transfer function is not easily described in words as it depends on the units of A_s and whether it is a torque/force that will drive it. However it must relate how some "force" applied generates some motion $z_s(x, y)$ with an amplitude A_s. Multiple `smotion` commands can be used at the same time on a component.

This motion can then be treated like any other, we can output them using:

```
xd name component s0
xd name component s1
xd name component s2
...
xd name component sN
```

where we reference the surface motion using the variables `s0`, `s1`,..., `sN`, which refer to the motions in the order that they are defined in the script. If the ordering of the `smotion` commands are changed then the motion that the `sN` refers to will change.

5.6.1 Parametric gain detector

FINESSE provides a detector to output the open loop transfer function from a motion back into itself. This is primarily used for modelling of parametric instabilities, of which

an example of will follow in the next section. As described in [Evans], the parametric gain $\mathcal{R}_m$ of a motion and its opto-mechanical interaction in a particular system is given by:

$$\mathcal{R}_m = \mathfrak{Re}\left[\frac{\Delta A_s}{A_s}\right] \tag{5.15}$$

If $\mathcal{R}_m > 1$ the motion is unstable and will exponentially increase with time, a behaviour that cannot be modelled explicitly with FINESSE. The real part of complex open loop transfer function $\frac{\Delta A_s}{A_s}$ can be outputted using the command,

```
pgaind name component motion
```

5.7 Example: parametric instabilities

Parametric instabilities here concern how a surface motion can postively or negatively feedback into itself due to optical coupling. Modelling this requires both reasonable models of the surface motions of a particular mirror which is determined using Finite-Element-Model (FEM) software and the optical system in which it is present, which is modelled by FINESSE. From the FEM model the surface height variations must be extracted and stored in a file formatted identically to that of a static surface map (See section 4.7.5).

This example is based on the example shown in [Evans] which aims to demonstrate how a particularly bad parametric instability can be present if the overlap between the beam shape and surface motion is large. The surface motion that is used is shown in figure 5.5, as can be seen it appears similar in shape to that of the TEM_{11}. As the resonance frequency of such surface modes can vary, this simulation sweeps the motion resonance to see if and where the motion might become unstable.

```
l l1 3530 0 n1 # power so that there is 1MW in the cavity
tem l1 0 0 0 0
tem l1 1 1 1 0

m m1 0.986 0.014 0 n1 n2
s s1 3994.5 n2 n3
m m2 0.99999 1e-5 0 n3 n4

cav c1 m1 n2 m2 n3

attr m1 Rc -1934
attr m2 Rc 2245

tf tf1 1 0 p 30k 1E7

smotion m2 surf_mod.map tf1
```

```
pgaind R m2 s0
yaxis re

# To plot the sidebands in the cavity
# uncomment these and remove oltfd above
#ad upper 1 1 $fs n2
#ad lower 1 1 $mfs n2
#yaxis abs

fsig l_sig l1 1 0 1

# The aim is scan the resonance frequency of the motion
# to see which might become unstable.
put tf1 fp1 $x1

xaxis l_sig f lin 20e3 50e3 1000

maxtem 2
```

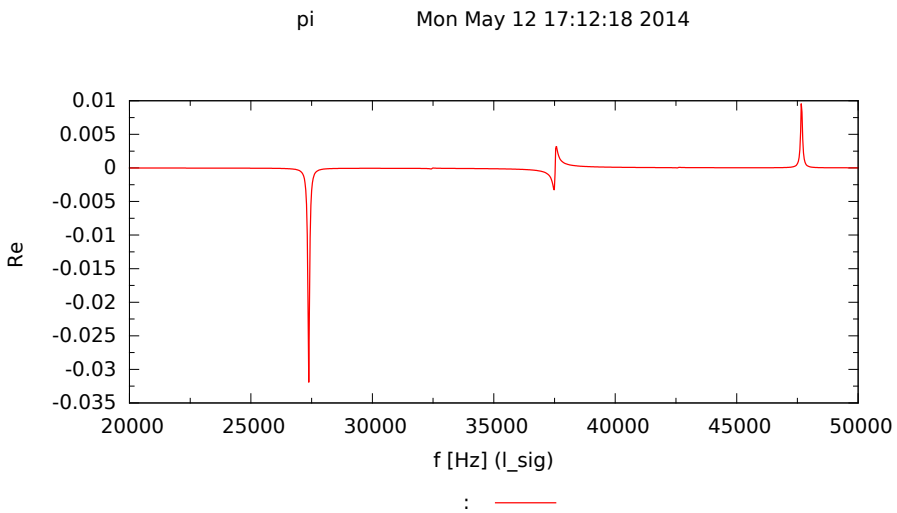


Figure 5.3: *Parametric gain of a surface motion with a large overlap with the TEM_{11} mode which is injected into the cavity. The resonance frequency of the motion is scanned and the we find $\mathcal{R}_m < 1$ so the motion does not become unstable.*

It can be see from figure 5.3 that $\mathcal{R}_m$ is always less than 1 thus not instable. We can see in figure 5.3 that when the upper sideband becomes resonant the mirror motion is damped, thus mechanical energy is extracted into the optical field. When the lower sideband becomes resonant energy is being fed into the motion of the mirror. This is an example of optical cooling.

It should also be noted that figure 5.3 differs slightly to what is found in [Evans]. This is due to the inclusion of all modes up to order 2, rather than just purely TEM_{11}, which is the cause for the slight difference around 37.5 kHz. The scaling also differs and this

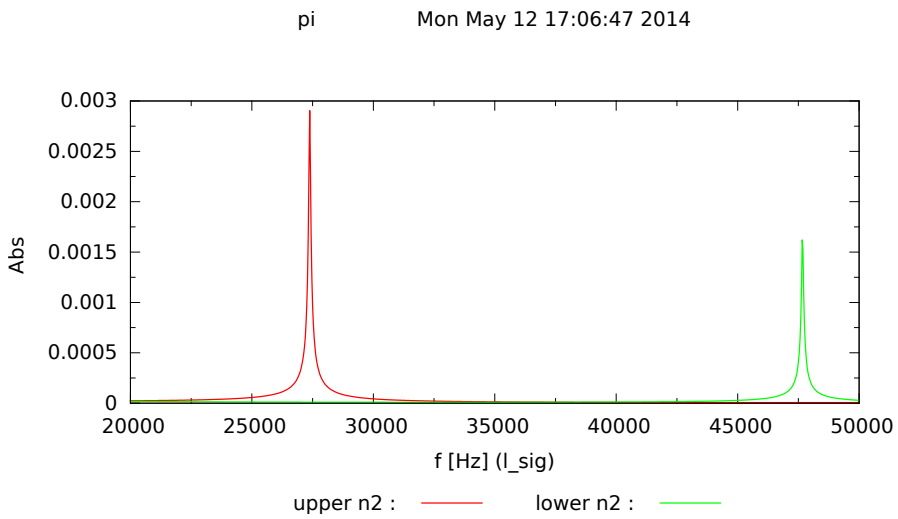


Figure 5.4: *Resonance of TEM_{11} signal sidebands in the cavity due to surface motion.*

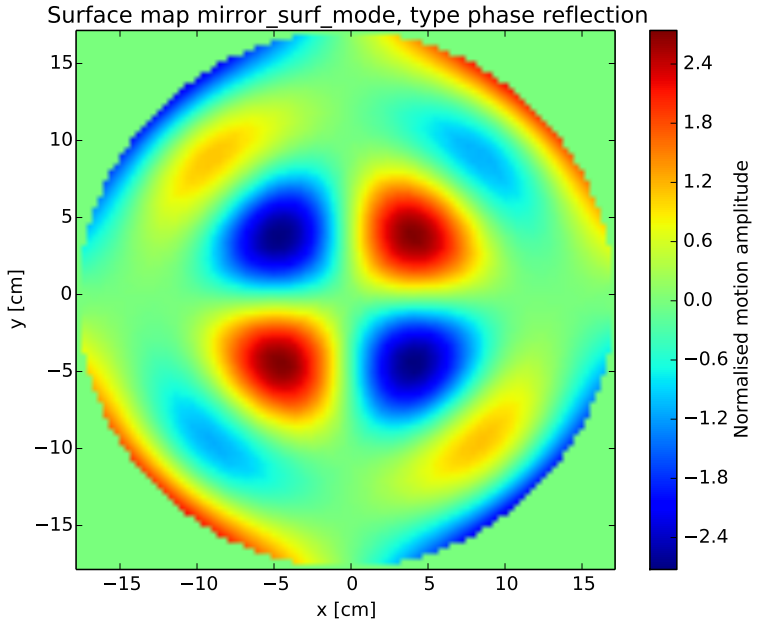


Figure 5.5: *Surface motion map*

due to the normalisation of the map data. It is stated in [Evans] that the function is normalised to equal the volume of the optical element, this was not done in this case.

Chapter 6

Quantum noise

From version 2.0 FINESSE has the ability to model quantum noise as described by the two-photon formalism [Caves], including the radiation pressure induced coupling between noise quadratures and the injection of squeezed vacuum. This feature is based on a computationally efficient method for evaluating the quantum noise for many fields (such as higher-order modes) and many open ports (e.g through optical losses).

6.1 New quantum noise modelling

The quantum noise computations are based on two-photon formalism [Caves]. This section introduces a description of quantum noise using two correlated photons, or a pair of symmetric sidebands, to model the variance of the phase and amplitude fluctuation of the light. These quantum noise sidebands are not dissimilar to signal sidebands as discussed previously. In fact they are solved using exactly the same interferometer matrix. The key difference between quantum noise and signal sidebands is that the signals are considered coherent whereas quantum noise sources are incoherent and their contribution to the total noise at an output must be computed accordingly.

FINESSE is often used to model one output signal of an optical system with many open ports, such as gravitational wave detectors. We have developed a algorithm which takes this into account, using a method to propagate all quantum noise sources in as few a steps as possible [Brown], by computing $\mathbb{V}_o$, the output covariance matrix of the quantum noise sidebands,

$$\mathbb{V}_o \quad = \quad \mathbb{M}_q^{-1}\mathbb{V}_i\mathbb{M}_q^{-1\,\dagger}. \tag{6.1}$$

Here $\mathbb{M}_q$ is the quantum noise sideband interferometer matrix and $\mathbb{V}_i$ is the input variances-covariance matrix of the quantum noise sidebands. $\mathbb{M}_q$ is a matrix that describes all optic to optic, optical to mechanical and mechanical to optical couplings for computing the transfer functions of noise to any node in the interferometer. Once $\mathbb{V}_o$ is known various outputs regarding the quantum noise can be computed such as photodiode noise, squeezing factors, amplitude and phase quadratures.

6.1.1 Sources of quantum noise

FINESSE by default includes all possible vacuum noise sources in any simulation which includes a quantum detector. Therefore, to model quantum noise you need only include the appropriate detectors as described below. The three noise sources currently implemented in FINESSE are:

- Open ports: An open node (connected to only one component) has vacuum state fluctuations injected through this node. Note that a 'dump' node is not treated as an open port.
- Optical losses: Any optical loss defined at a component such as a mirror or beamsplitter will inject a defined amount of vacuum noise back into the respective nodes of the component.
- Squeezed input: A squeezing input will inject squeezed vacuum into the connected node.

To see which quantum noise sources are present in a particular simulation you can use the command,

```
printnoises
```

(this will only work if you have specified one of the quantum noise detectors mentioned in the next section) producing text output such as::

```
-- Quantum noise inputs --
  |- sq1        - SQUEEZED INPUT Node nfc1
  |- mfc1       - OPEN PORT Node nfc2
  |- mfc2       - LOSS Node nfc5
  |- mfc2       - LOSS Node nfc6
  |- mfc2       - OPEN PORT Node nfc7
  |- mfc2       - LOSS Node nfc7
  |- mfc2       - OPEN PORT Node nfc8
  |- mfc2       - LOSS Node nfc8
```

Here the first column is the component where the noise is injected, the second states what type of noise and the node at which it is incoming. You can also choose to model specific components and their noise sources if you wish using the command:

```
vacuum component1 component2 ...
```

where each component you want to include noise from is listed. Doing this allows you to model how specific noise will affect certain outputs or find out which is the dominant noise source for example. When the `vacuum` command is used **only the specified components will inject vacuum**, all other losses, or open ports or squeezer components will not.

It should also be noted that `dump` nodes are ignored completely by FINESSE, thus dump nodes do not act like open ports. For noise to be injected into the the system you must give the node a name.

Setting the noise frequency

The frequency at which quantum noise sidebands are computed at the same frequency as signal sidebands. To specify which frequency the noise is computed at the signal sideband frequency must be set with the `fsig` command for both signal and noise computations. However in some simulations you may not need to actually model some signal being applied and are just interested in computing noises. In this case you can use the shortend version, where you specify just a name for the frequency as a handle and the initial value,

```
fsig name f
```

You can then sweep the frequency using an `xaxis` by targetting `name` and the `f` parameter.

Squeezed light injection

Squeezed light can be injected into any node using the new squeezing component. This behaves like a light input but it does not inject any carrier fields. You need to state how many decibels of squeezing you need and how the squeezed state is rotated in degrees. You can use the quantum noise detectors `qnoised`, `qd` or `sd` (which are described next) to see the effect. The relation between the squeezing factor we set in dB r_{db} and the squeezing factor used for calculations r is:

$$r_{db} = 20\, r\, \log_{10}(e). \tag{6.2}$$

To note, setting a squeezing of 20 dB in FINESSE will result in reduction in the amplitude spectral density of the noise up to a factor of 10.

6.1.2 Quantum detectors

The key aspects of the new quantum noise feature are the new quantum detectors. These allow you to extract information about the quantum noise of the optical system. There are currently two types of detectors: those to detect the noise of the photocurrent produced at a photodiode, for example to produce a quantum-noise limited sensitivity curve, and those for reading out quantum noise around a specified carrier frequency.

Photodiode noise: qnoised and qshot

The quantity we are most interested in computing is the photocurrent noise of a particular photodiode output, particularly to compute sensitivities or noise-to-signal ratios (see 6.1.2). The two available detectors for computing the noise of a photodiode output are called `qnoised` and `qshot`. Both detectors commands follow closely the photodiode commands. Like the normal `pd` detectors, the `qnoised` and `qshot` accept up to 5 demodulations. The demodulation of a photocurrent mixes different components of the quantum noise present in the optical fields, which must be correctly taken into account as outlined

in [Harms]. The difference between `qnoised` and `qshot` is that `qshot` assumes only shot noise (or a pure vacuum state) is present in the detected fields, thus this detectors will not see any effects from squeezing or radiation pressure. However, it required less computing resources and can be used if such effects are not of interest. The `qnoised` detector returns the proper quantum noise of the detected fields and thus takes into account both radiation pressure effects and squeezing. You may chose to compre the outputs of both of these detectors to highlight the impacts of squeezing or radiation pressure effects. For `qnoised` to see any radiation pressure or squeezing effects the final demodulation frequency must be set to the *signal frequency* specified with `fsig`. FINESSE takes all the carrier fields and applies sidebands at the `fsig` frequency to all of them, this is the frequencies used to compute the radiation pressure and quantum noise effects. For example, for a photodiode that has one demodulation at frequency `f` with phase `phi` and we wish to see radiation pressure or squeezing effects we would use the code:

```
qnoised name 2 f phi $fs node
```

where the variable `$fs` refers to the current positive signal frequency of the simulation (see section 3.1.2). This detector outputs an amplitude spectral density by default, see 6.1.2 for more information. Currently (FINESSE 2.0) quantum noise detectors do not use the quantum efficiency setting given in the `kat.ini` file, this feature will likely be added in an upcoming version. In the meantime the same effect can be reproduced using a lossy mirror with zero reflectivity in front of the detector.

It should be noted both `qnoised` and `qshot` cannot use the `max` helper keyword for a demodulation phase due to the implementation of the algorithm. Thus numerical values have to be specified for the demodulation phases for the correct noise values to be computed.

Sensitivity (noise-to-signal): qnoisedS and qshotS

A typical usage of the quantum noise detectors is to generate so-called sensitivity plots, which are effectively noise-to-signal ratios, where we apply some signal to the interferometer using `fsig` and then measure how that transfers to some photodiode output. This can be achieved by using both a `pd` for computing the signal and `qnoised` or `qshot` for the noise, and then using `func` to divide the two. To simplify that process you can use the 'S' or 'sensitivity' versions of both `qnoised` and `qshot`. For example: be shown with,

```
qnoisedS name 2 f1 phi1 $fs max node
```

Note, that in this case the `max` keyword can be used, because a signal computation is included: Internally the signal part is computed first, whereby the optimum demodulation phase for the signal is found, this phase is then set automatically for the noise computation providing a correct noise output. You should also keep in mind that when using this you may require some scaling of the signal transfer function as described in section 6.2 depending on the type of the applied signals.

Photodiode noise output scaling

By default in FINESSE the detectors `qnoised`, `qshot` and `shot` will return amplitude spectral densities (ASD) of the noise, as this is what is commonly plotted in sensitivity plots. This behaviour can changed using the `quantum_scaling` settings in the `kat.ini` file. Note when comparing results from other people or different machines, large differences noise levels could well be due to this setting! The options are:

```
# 1 = Power spectral density
# 2 = Power spectral density in units of hf
# 3 = Amplitude spectral density (default)
# 4 = Amplitude spectral density in units of sqrt(hf)
```

Note that you can choose to not include the hf factors, this provides a numerically simpler values for some comparisons with theory.

Squeezing detector: measuring the vacuum

The squeezing detector `sd`:

```
sd name carrier_frequency [n m] node[*]
```

can be used to determine whether a carrier field is squeezed or not. The output describes the noise ellipse for a particular carrier (for the given TEM_{nm} mode). The detector outputs a complex number whose magnitude is the squeezing level in dB r_{db}. The 'squeezing factor' r is related to r_{dB} by:

$$r_{db} = 20\, r\, \log_{10}(e). \tag{6.3}$$

The phase of the complex output is ϕ the squeezing angle in degrees (use the `yaxis` command to output phase information). When $\phi = 0, 180$ and $r \neq 0$ then the carrier is amplitude squeezed, when $\phi = \pm 90$ and $r \neq 0$, the carrier is phase squeezed.

This detector only tells you information about a specific carrier field defined by the frequency offset value (and the mode indicies).

Quadrature detector: debugging the squeezed state

The quadrature detector, `qd`:

```
qd name carrier_frequency phase node[*]
```

allows you to extract a particular noise quadrature of a single carrier field. With this you can for example plot the amplitude and phase quadratures of a field given in units of hf, where f is the frequency of the carrier and h Planck's constant. A pure vacuum state would output a value of '1' for both quadratures, any deviation from this shows squeezing or an increase in the noise due to losses. The crucial part in using this detector is selecting the correct quadrature angle; an angle of 0 will output the carrier fields

amplitude quadrature, 90 will output its phase quadrature noise. Of coure you can also choose any angle between 0 and 360.

6.1.3 Example: unbalanced quantum noise homodyne detection

An unbalanced homodyne detector setup is a good way to understand how the various detectors work. The setup consists of a laser, a squeezed source, a 50:50 beamsplitter and a photodiode. We mix the squeezed and laser fields and measure the output at the photodiode.

We mix a 10dB squeezed source and a 1W laser at the beamsplitter and measure the result at the photodiode, varying the phase of the laser field.

```
l l1 1 0 n1
s s1 0 n1 n2
sq sq1 0 10 0 n3
s s2 0 n3 n4
bs bs1 0.5 0.5 0 45 n4 nout1 nout2 n2

# Setting the frequency at which we compute the quantum noise
# and signals. We aren't applying any signal, we just need
# to set the frequency value and give it a name.
fsig noise 1

# output the pure shot noise along with the qnoised detector
# measuring the effects of the squeezing
qnoised sqzd_noise 1 $fs 0 nout1
qshot   shot_noise 1 $fs 0 nout1

xaxis l1 phase lin -90 90 360
```

Figure 6.1 shows the output of the script: the shot noise output is flat whereas the `qnoised` detector output shows and increase or reducing below the shot noise, depending on the phase. You should also see here that you must be careful to ensure that any squeezed field and laser fields have the correct phase and squeezing angle to ensure that you have the effect you desire (try for example to change the length of `s1` and see what happens).

With the second file in this example, the `qd` detectors are used to extract the quadrature values for the laser field.

```
l l1 1 0 n1
s s1 0 n1 n2
sq sq1 0 10 0 n3
s s2 0 n3 n4
bs bs1 0.5 0.5 0 45 n4 nout1 nout2 n2
```

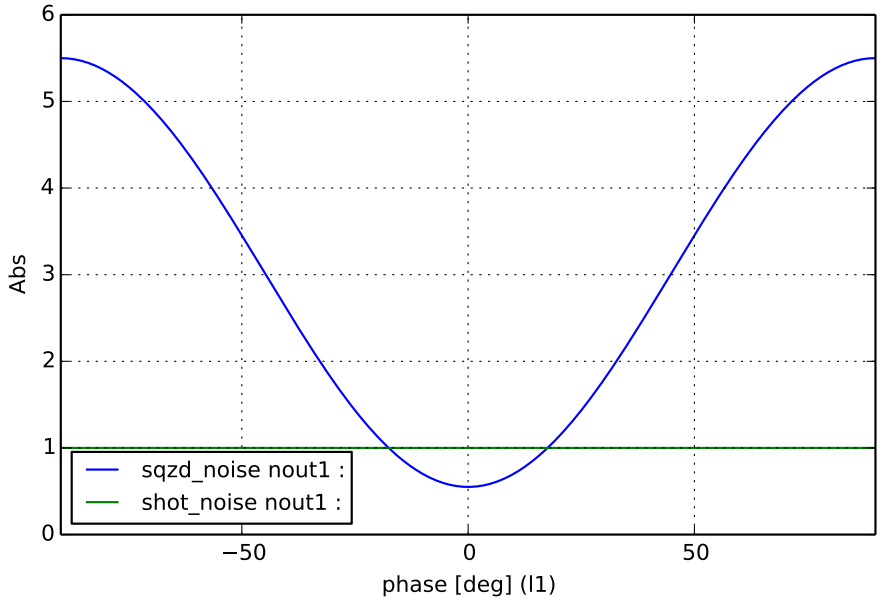


Figure 6.1: *`qnoised` and `qshot` detector outputs for a homodyne setup. The laser phase is varied to show squeezing and anti-squeezing can be seen. The noise scaling for this plot has been set to 'Power spectral density in units of hf' via the `quantum_scaling` setting in the 'kat.ini' file.*

```
# defining a 'signal' or noise frequency
fsig noise 1

# setup quadrature detectors to output both the
# amplitude and phase quadratures
qd qdA 0 0 nout1
qd qdP 0 90 nout1

xaxis l1 phase lin -90 90 360
```

In figure 6.2 the output is shown and we can clearly see how the amplitude and phase quadratures of the laser field are varying.

Finally, the `sd` detector can be used to extract how much squeezing, and at what angle, we have. Note for this example we have changed the squeezing angle of the input source by 45 degrees, just for something to measure.

```
l l1 1 0 n1
s s1 0 n1 n2
sq sq1 0 10 45 n3
s s2 0 n3 n4
bs bs1 0.5 0.5 0 45 n4 nout1 nout2 n2
```

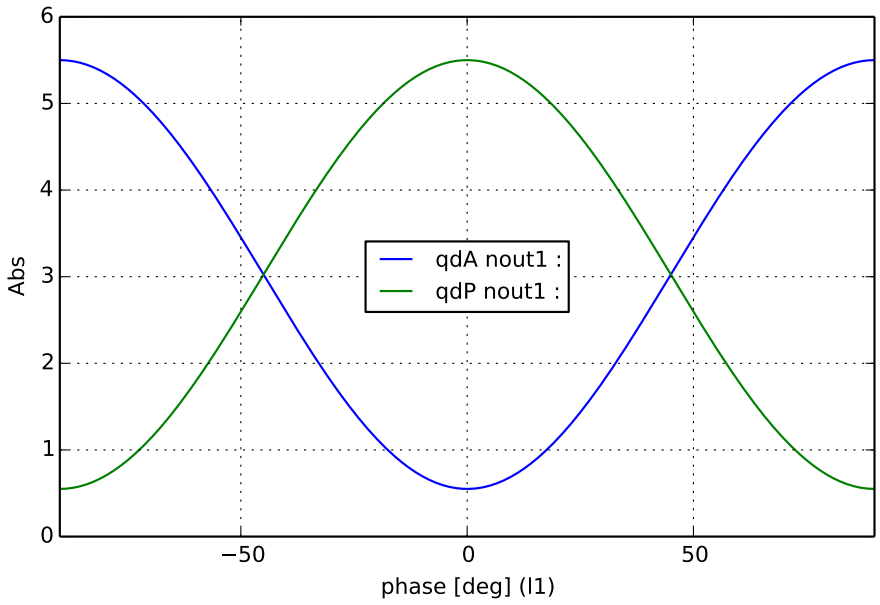


Figure 6.2: *qd detector outputs shows how both the amplitude and phase quadratures vary due to a change in the laser phase.*

```
fsig noise 1

sd sd1 0 nout1

xaxis l1 phase lin -90 90 360
yaxis abs:deg
```

The results are shown in figure 6.3. This is a rather boring plot, but we can see that the noise ellipse has a squeezing factor $r_{db} = 5$dB and is rotated by an angle of $\phi = 45$ degrees.

6.1.4 Example: dual-recycled, quantum-noise limited sensitivity

This next example brings together many of the new features for computing the quantum-noise limiting sensitivity of a dual-recycled michelson interferometer. The model is loosely based on the Advanced LIGO design file and thus we can expect to see the peak sensitivity around 100 Hz at a sensitivity of about $10^{-23}/\sqrt{Hz}$. The file will firstly setup all the various optical cavities (using a plane waves model). It then proceeds to suspend the arm cavity mirrors whilst setting the mechanical suspension transfer functions to a simple pendulum with a resonance at 1 Hz. Next, a gravitational wave signal is injected as a modulation to both arm 'spaces', out of phase by 180 degrees. Lastly we use the `qnoisedS` and `qshotS` detectors to output the noise-to-signal ratio, or the sensitivity.

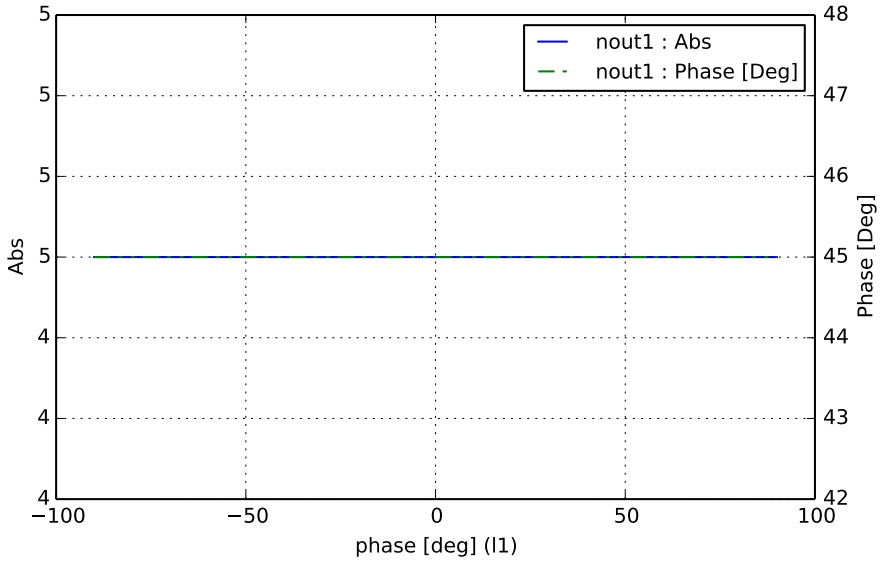


Figure 6.3: *Example output of an* `sd` *detector in a homodyne setup.*

```
# File based on aLIGO reference design, DCC number M060056

# Laser input
l l1 $Pin 0 nin
s s1 0 nin nprc1
# Power recycling mirror
m1 prm $prmT 37.5u 90 nprc1 nprc2
s  prc $lprc nprc2 nbsin
# Central beamsplitter
bs bs1 .5 .5 0 45 nbsin n0y n0x nbsout

# X-arm
s ichx $lmichx n0x n1x
m1 itmx $itmT 37.5u 90 n1x n2x
s armx $Larm n2x n3x
m1 etmx 5u 37.5u 89.999875 n3x n4x
attr itmx mass $Mtm zmech sus1
attr etmx mass $Mtm zmech sus1

# Y-arm
s  ichy $lmichy n0y n1y
m1 itmy $itmT 37.5u $michy_phi n1y n2y
s  army $Larm n2y n3y
m1 etmy 5u 37.5u 0.000125 n3y n4y
attr itmy mass $Mtm zmech sus1
attr etmy mass $Mtm zmech sus1

# Signal recycling mirror
```

```
s  src $lsrc nbsout nsrc1
m1 srm $srmT 37.5u $srm_phi nsrc1 nsrc2

# Force-to-position transfer function for longitudinal
# motions of test masses
tf sus1 1 0 p $mech_fres $mech_Q
const mech_fres 1  # 9 sus-thermal spike
const mech_Q    1M # Guess for suspension Q factor
# DC readout: 100mW = michy_phi 0.07 _or_ darm_phi .00025
const michy_phi 0
const darm_phi  .00025

const Larm 3995
const itmT 0.014
const srmT 0.2
const prmT 0.03
const Pin  125
const Mtm  40
const srm_phi -90
const lmichx 4.5
const lmichy 4.45
const lprc   53
const lsrc   50.525

# A squeezed source could be injected into the dark port
sq sq1 0 0 90 nsrc2

# Differentially modulate the arm lengths
fsig darm  armx 1 0
fsig darm2 army 1 180

# Output the full quantum noise limited sensitivity
qnoisedS NSR_with_RP    1 $fs nsrc2
# Output just the shot noise limited sensitivity
qshotS   NSR_without_RP 1 $fs nsrc2

# We could also display the quantum noise and the signal
# separately by uncommenting these two lines.
# qnoised noise    $fs nsrc2
# pd1     signal 1 $fs nsrc2

xaxis darm f log 5 5k 1000
yaxis log abs
```

The final plot is shown in figure 6.4. Here we can see the both the `qnoised` and `qshot` agree at high frequency, which they should as both model shot noise correctly. At low frequencies we see that they differ as `qshot` does not detect any radiation pressure effects.

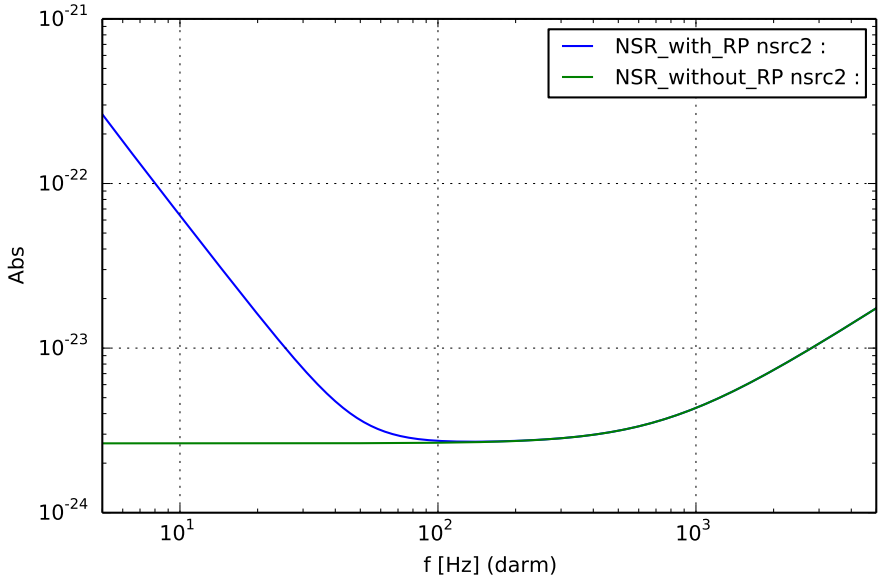


Figure 6.4: *A dual-recycled michelson quantum noise limited sensitivty plot*

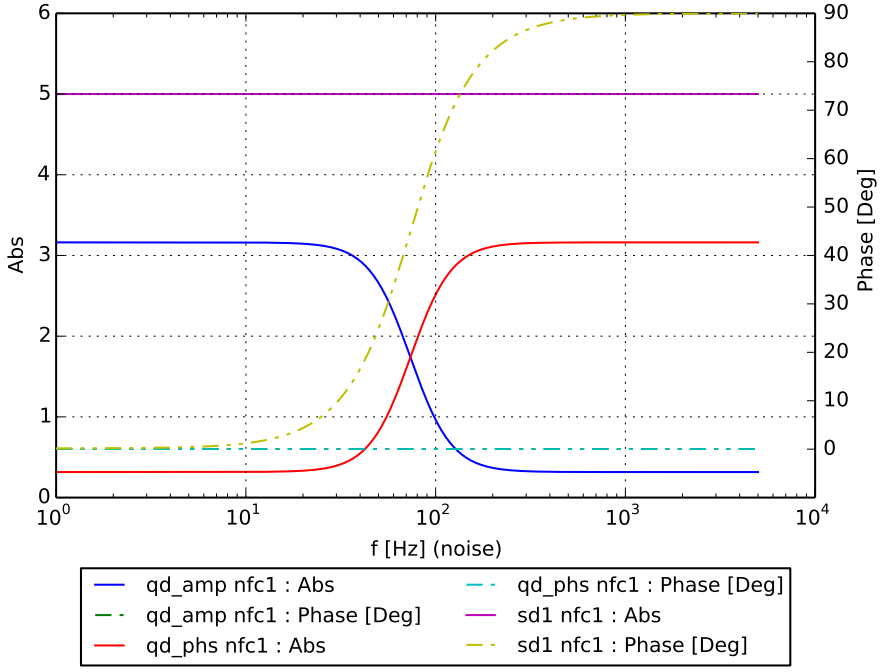


Figure 6.5: *Perfect mirror filter cavity rotating the noise ellipse 90 degrees to get frequency dependent squeezing*

6.1.5 Example: a filter cavity, or how to rotate a squeezed state

This example aims to show how filter cavities can be modelled or how such cavities can rotate a squeezing ellipse [Kimble]. The setup includes a 16 m long cavity with a perfectly reflective end mirror and 70 ppm transmission for the input mirror. The end mirror is then slightly detuned and we reflect a squeezed input (5 dB squeezing) of the cavity.

```
sq sq1 0 5 90 nin
s s1 0 nin nfc1

m1 mfc1 70u 0 0 nfc1 nfc2
s sfc1 16 nfc2 nfc3
m1 mfc2 0 0u -0.001 nfc3 nfc4

qd qd_amp 0 0 nfc1
qd qd_phs 0 90 nfc1

sd sd1 0 nfc1
fsig noise 1

xaxis noise f log 1 5000 10000
yaxis abs:deg
printnoises
```

We then use both the `qd` and `sd` detectors to see what is happening. Figure 6.5 shows the results. Firstly we see that the amplitude and phase quadratures rotate by 90 at around 75 Hz, at low frequency we have phase squeezing and at high we have amplitude squeezing, in this particular case – you can simply adjust the squeezing angle at the input to flip this behaviour around. The `sd` detector also shows that for the whole bandwidth we have 5 dB of squeezing, so it never degrades, and that we do in fact have a rotation in the noise ellipse of 90 degrees.

But what happens if we include some losses? Let's add 100 ppm losses to the end mirror, the result is shown in figure 6.6.

```
sq sq1 0 5 90 nin
s s1 0 nin nfc1

m1 mfc1 70u 0 0 nfc1 nfc2
s sfc1 16 nfc2 nfc3
m1 mfc2 0 10u -0.001 nfc3 nfc4

qd qd_amp 0 0 nfc1
qd qd_phs 0 90 nfc1
```

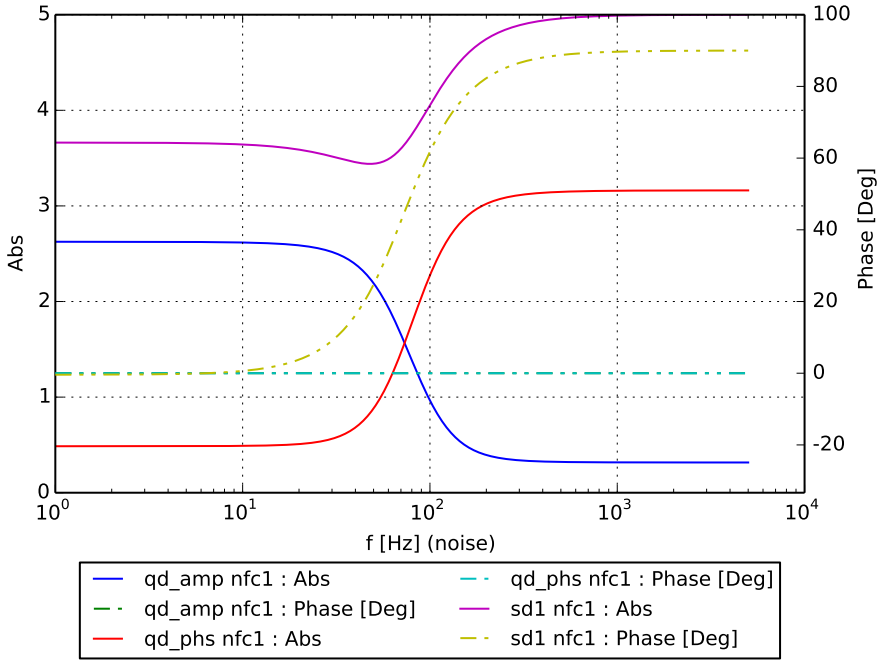


Figure 6.6: *Lossy filter cavity behaviour showing squeezing degradtion at low frequencies due to a lossy end mirror.*

```
sd sd1 0 nfc1
fsig noise 1

xaxis noise f log 1 5000 10000
yaxis abs:deg
printnoises
```

We now see that the squeezing is being degraded at low frequency due to the vacuum noise entering the cavity from the lossy end mirror. This only happens at low frequency when those vacuum fluctuations are within the cavity line width.

6.2 Shot-noise-limited sensitivity (before version 2.0)

This section outlines the older shot noise computations that were available before version 2.0, it does however contain useful information regarding correct scaling of sensitivity limits and transfer functions.

This section gives a detailed description on how FINESSE computes shot-noise using the `shot` command and the respective linear spectral density that is often called sensitivity. In has been shown to be difficult to remember all factors of 2 correctly, and thus a step by step explanation for a simple example system is provided here.

Appendix B further provides a comparison between different methods of computing the shot noise with FINESSE: analytically from the DC power, using the `shot` command and using the new `qshot` command, which can correctly compute the shotnoise for heterodyne detection schemes. The appendix is based on a dedicated effort by the GEO collaboration in 2007 to understand and verify the shot noise level limiting the GEO sensitivity at high frequencies.

The shot noise computation is based on the Schottky formula for the (single-sided) power spectral density of the fluctuation of the photocurrent for a given mean current $\bar{I}$:

$$S_I(f) = 2\,e\,\bar{I}, \tag{6.4}$$

with e the electron charge. Here $S_X(f)$ denotes the single-sided power spectral density of X over the Fourier frequency f.

The link between (mean) photocurrent $\bar{I}$ and (mean) light power $\bar{P}$ is given by the relation:

$$\bar{I} = eN = \frac{e\ \eta\ \lambda}{\hbar 2\pi c}\bar{P}, \tag{6.5}$$

with N as the number of photons and η the quantum efficency of the diode. Intead of Planck's constant we write $\hbar\,2\,\pi$ to avoid confusion with the typical use of $h(t)$ for the strain of a gravitational wave.

Thus we can now give a power spectral density for the fluctuations of the photocurrent. The conversion between Watts in Ampere is defined by the constant C_2 in the relation $\bar{P} = C_2\,\bar{I}$. Thus the power spectral densities must be converted as $S_P(f) = C_2^2 S_P(f)$. With the relation $S_I(f) = C_1\,\bar{I}$ we can then write:

$$S_P(f) = C_2^2 S_I(f) = C_2^2\ C_1\,\bar{I} = C_2^2\ C_1\ \bar{P}/C_2 = 2\,\frac{2\pi\,\hbar\,c}{\lambda}\bar{P}. \tag{6.6}$$

6.2.1 Simple Michelson interferometer on a half fringe

A simple example for computing a shot-noise-limited sensitivity is a Michelson interferometer held on a half fringe. In this case no modulation sidebands are required to obtain the output signal. In the following examples we will use:

- an input laser power of $P_0 = 1\,\mathrm{W}$,
- the quantum efficiency is `qeff` $= \eta = 1$,
- a symmetric beam splitter with $R = T = 0.5$ (angle of incidence in FINESSE is set to 0 deg),
- we denote the interferometer outputs (arbitrarily following the GEO 600 layout) as 'north', 'east', 'south' and 'west', with the input port being in the west and the light reflected by the beam splitter entering the north arm,
- the Michelson interferometer arm lengths are chosen as $L_{\mathrm{north}} = 1201\,\mathrm{m}$ and $L_{\mathrm{east}} = 1200\,\mathrm{m}$.

The noise amplitude

The Michelson interferometer is set to a half fringe by detuning the beam splitter by 22.5 degrees so that the power in both the west and south output ports is 0.5 W. From Equation 6.6 we expect a value of:

$$S_P(f) = \frac{6.6262\,10^{-34}\ 299792458}{1064\,10^{-9}}\,\mathrm{W^2/Hz} = 1.867\,10^{-19}\,\mathrm{W^2/Hz}, \tag{6.7}$$

or as a linear spectral density $\sqrt{S_P} = 4.321\,10^{-10}\,\mathrm{W}/\sqrt{\mathrm{Hz}}$. FINESSE returns the same value if the `shot` detector is used.

The signal amplitude

The 'signal' in this example will be a differential motion of the end mirrors. In order to compute the signals' amplitude in the photodiode we can compute the transfer function of the mirror motion to the photodiode. A motion of the end mirrors will modulate the reflected light in phase.

In the following we set all macroscopic lengths to be multiples of the wavelength; the signal frequencies are assumed to be very small so that the phase evolution for the carrier and the signal sidebands due to the free propagation through the interferometer arms can be considered equal.

The light fields entering the arms are given by:

$$E_{\mathrm{N\,in}} = \frac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\frac{\pi}{4}\right), \tag{6.8}$$

$$E_{\mathrm{E\,in}} = \frac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\frac{\pi}{2}\right). \tag{6.9}$$

Sidebands are created upon reflection on the end mirrors. The phase of the modulation is set to be 0° at the north mirror and 180° at the east mirror. The phase of the sidebands is given by Equation 3.103:

$$\varphi_{\mathrm{sb}} \;=\; \varphi_{\mathrm{c}} + \frac{\pi}{2} \pm \varphi_{\mathrm{s}} - (k_{\mathrm{c}} + k_{\mathrm{sb}})\; x_{\mathrm{t}}. \tag{6.10}$$

The light reflected by the end mirrors (with $r = 1$) can then be written as:

$$\begin{aligned} E_{\mathrm{N}} &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{4}\right) \left(1 + a_s \exp\left(\mathrm{i}\,\omega_s t + \tfrac{\pi}{2}\right) + a_s \exp\left(-\mathrm{i}\,\omega_s t - \tfrac{\pi}{2}\right)\right) \\ &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{4}\right) \left(1 + a_s \mathrm{i}\,\left(\exp\left(\mathrm{i}\,\omega_s t\right) + \exp\left(-\mathrm{i}\,\omega_s t\right)\right)\right) \\ &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{4}\right) \left(1 + 2a_s \mathrm{i}\,\cos\left(\omega_s t\right)\right), \\ E_{\mathrm{E}} &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right) \left(1 + a_s \exp\left(\mathrm{i}\,\omega_s t + \tfrac{3\pi}{2}\right) + a_s \exp\left(-\left(\mathrm{i}\,\omega_s t + \tfrac{\pi}{2}\right)\right)\right) \\ &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right) \left(1 - 2a_s \mathrm{i}\,\cos\left(\omega_s t\right)\right), \end{aligned} \tag{6.11}$$

with ω_s as the signal frequency and a_s the amplitude of the mirror motion (given in radians, with 2π referring to a position change of the mirror of λ). For a computation of the transfer function in our example we later set $a_s = 1$ while at the same time we use the approximations for $a_s \ll 1$.

At the beam splitter the reflected fields will be superimposed, in the south port we get:

$$\begin{aligned} E_\mathrm{S} &= \tfrac{1}{\sqrt{2}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right) E_\mathrm{N} + \tfrac{1}{\sqrt{2}} \exp\left(-\mathrm{i}\,\tfrac{\pi}{4}\right) E_\mathrm{E} \\ &= \tfrac{1}{2} E_\mathrm{in} \exp\left(\mathrm{i}\,\tfrac{\pi}{4}\right) \left(1 + \mathrm{i} - 2a_s \mathrm{i} \cos\left(\omega_s t\right) - 2a_s \cos\left(\omega_s t\right)\right) \\ &= \tfrac{1}{2} E_\mathrm{in} \exp\left(\mathrm{i}\,\tfrac{\pi}{4}\right) \left(1 + i\right) \left(1 - 2a_s \cos\left(\omega_s t\right)\right) \\ &= \tfrac{\mathrm{i}}{\sqrt{2}} E_\mathrm{in} \left(1 - 2a_s \cos\left(\omega_s t\right)\right). \end{aligned} \tag{6.12}$$

With $|E_\mathrm{in}|^2 = P_0 = 1\,\mathrm{W}$ the power detected by the diode in each output port is thus:

$$\begin{aligned} I_\mathrm{out} &= \tfrac{1}{2} P_0 \left(1 + 4a_s^2 \cos^2\left(\omega_s t\right) - 4a_s \cos\left(\omega_s t\right)\right) \\ &= \tfrac{1}{2} \left(1 + 4a_s^2 \cos^2\left(\omega_s t\right) - 4a_s \cos\left(\omega_s t\right)\right)\ [\mathrm{W}]. \end{aligned} \tag{6.13}$$

The power in the signal sidebands can be neglected so that the DC power in both output ports is given as $P_0/2 = 0.5\,\mathrm{W}$.

The signal amplitude is given by $2a_s P_0$. In Finesse to get the signal amplitude we demodulate at the signal frequency, i.e. we multiply the output by $\cos \omega_s t$ and take the DC part of the resulting sum:

$$\begin{aligned} I_\mathrm{demod} &= 2a_s P_0 \cos^2\left(\omega_s t\right) + O(\omega) + O(3\omega) \\ &= a_s P_0 + a_s P_0 \cos\left(2\omega_s t\right) + O(\omega) + O(3\omega). \end{aligned} \tag{6.14}$$

The signal amplitude is thus given by $a_s P_0$. By default a demodulation in Finesse is understood as a multiplication with a cosine and thus reduces the signal size by a factor of 2, with the exception that in the case of the transfer function this is not wanted for the demodulation at the signal frequency.

With Finesse it is simple to compute a transfer function for a differential displacement of the end mirrors into the detector output. For example, with the commands

```
fsig sig1 mE 1 0
fsig sig2 mN 1 180
pd1 south1 1 max n10
```

we compute the signal transfer function at 1 Hz. The Finesse output is: 2 W/rad. To obtain the more useful units W/m we have to multiply by $2\pi/\lambda$.

It is important to understand which lengths we refer to with this transfer function. Due to the fact that Finesse can compute more general optical configurations than a Michelson interferometer, the amplitude of the transfer function amplitude refers to the motion of each single mirror. For example, a transfer function amplitude of 1 W/m means that the

output power changes by one 1 nW when the east mirror is moved by 1 nm and the north mirror by -1 nm. If we want to compute the transfer function referring to the differential displacement $\Delta L = L_{\text{north}} - L_{\text{east}}$ we have to divide the transfer function by a factor of two. Thus we get:

$$T_{\Delta L \to P} = \frac{2\pi P_0}{\lambda} = \frac{2\pi}{1064\,10^{-9}}\,\text{W/m}. \tag{6.15}$$

If, in fact, the transfer function is to be given with respect to an optical path length difference $\Delta L'$ one has to divide by another factor of two:

$$T_{\Delta L' \to P} = \frac{\pi P_0}{\lambda} = \frac{\pi}{1064\,10^{-9}}\,\text{W/m}. \tag{6.16}$$

Apparent length noise

Now we can compute the apparent end-mirror motion (measured as an optical path length difference) due to shot noise dividing the noise spectral density from Equation 6.6 by the transfer function:

$$\sqrt{S_{\Delta L'}(f)} = \frac{\sqrt{\frac{2\pi\,\hbar\,c}{\lambda} P_0}}{\frac{\pi P_0}{\lambda}} = \sqrt{2\frac{\hbar c \lambda}{\pi P_0}} = 1.46310^{-16}\,\text{m}/\sqrt{\text{Hz}}. \tag{6.17}$$

As expected we get exactly 0.25 times this value (i.e. $3.658504314e-17$) using FINESSE with:

```
fsig sig1 mE 1 0
fsig sig2 mN 1 180
pdS1 south1 1 max n10
scale meter
```

With two detectors, one in the west and one in the south port we can expect to have a better sensitivity by a factor of $\sqrt{2}$. The detected signal is twice the signal detected in a single port. Also, the detected total DC power increases by a factor of 2. Thus we expect the signals-to-shot noise to increase by a factor of $\sqrt{2}$. For our example we get $\sqrt{S_{\Delta L}(f)} = \sqrt{\frac{\hbar c \lambda}{\pi P_0}} = 1.03410^{-16}\,\text{m}/\sqrt{\text{Hz}}$.

6.2.2 Simple Michelson interferometer on a dark fringe

The following section demonstrates how to compute the shot-noise-limited sensitivity for a Michelson interferometer on the dark fringe. However, since phase modulation is employed the results based on the Schottky formula alone are not correct [Meers, Niebauer]. The following calculation is meant as an exercise to show that – when the effects of the modulation are neglected – the shot-noise-linited sensitivity at the dark fringe is exactly as for a Michelson interferometer on a half fringe with two detectors.

Again we start with the light fields entering the arms which are now given by:

$$\begin{aligned} E_{\mathrm{N\,in}} &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right), \quad \text{and} \\ E_{\mathrm{E\,in}} &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right). \end{aligned} \tag{6.18}$$

The light reflected by the end mirrors (with $r = 1$) can then be written as:

$$\begin{aligned} E_{\mathrm{N}} &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right) \left(1 + a_s \mathrm{i}\, \left(\exp\left(\mathrm{i}\,\omega_s t\right) + a_s \exp\left(-\mathrm{i}\,\omega_s t\right)\right)\right) \\ &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \left(\mathrm{i}\, - 2a_s \cos\left(\omega_s t\right)\right), \\ E_{\mathrm{E}} &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right) \left(1 - a_s \mathrm{i}\, \left(\exp\left(\mathrm{i}\,\omega_s t\right) + a_s \exp\left(-\mathrm{i}\,\omega_s t\right)\right)\right) \\ &= \tfrac{1}{\sqrt{2}} E_{\mathrm{in}} \left(\mathrm{i}\, + 2a_s \cos\left(\omega_s t\right)\right). \end{aligned} \tag{6.19}$$

And hence in the south port we get:

$$\begin{aligned} E_{\mathrm{S}} &= \tfrac{1}{\sqrt{2}} \exp\left(\mathrm{i}\,\tfrac{\pi}{2}\right) E_{\mathrm{N}} + \tfrac{1}{\sqrt{2}} \exp\left(-\mathrm{i}\,\tfrac{\pi}{2}\right) E_{\mathrm{E}} \\ &= -2\mathrm{i}\, E_{\mathrm{in}} a_s \cos\left(\omega_s t\right). \end{aligned} \tag{6.20}$$

The photocurrent generated by this output field does not contain a signal at the frequency ω_s. Such a signal component can be produced with a modulation-demodulation technique. In this example we can simply add another pair of phase modulation sidebands and ignore how these are exactly generated. Let us assume we have symmetric modulation sidebands at ω_m reaching the photodiode in the south port. We get:

$$E_{\mathrm{S}} = \mathrm{i}\, b \left(\exp\left(\mathrm{i}\,\omega_m t\right) + \exp\left(-\mathrm{i}\,\omega_m t\right)\right) - 2\mathrm{i}\, E_{\mathrm{in}} a_s \cos\left(\omega_s t\right) \tag{6.21}$$

$$= 2\mathrm{i}\, b \cos\left(\omega_m t\right) - 2\mathrm{i}\, E_{\mathrm{in}} a_s \cos\left(\omega_s t\right). \tag{6.22}$$

The power detected by the diode in each output port is thus:

$$I_{\mathrm{out}} = 4b^2 \cos^2\left(\omega_m t\right) + 4a_s^2 E_{\mathrm{in}}^2 \cos^2\left(\omega_s t\right) - 8ba_s E_{\mathrm{in}} \cos\left(\omega_m t\right) \cos\left(\omega_s t\right) \tag{6.23}$$

$$= 2b^2 + 2a_s^2 P_0 - 8ba_s \sqrt{P_0} \cos\left(\omega_m t\right) \cos\left(\omega_s t\right) + O(2\omega_m) + O(2\omega_s). \tag{6.24}$$

$$\tag{6.25}$$

A demodulation at ω_m gives a signal amplitude proportional to $4ba_s\sqrt{P_0}$.

An example calculation can be done with a modulation at 10 MHz:

```
mod eom1 10M .1 1 pm n2 n3
```

Then we first check the field amplitudes and the DC power in the output port (at node `n10`) with:

```
pd dc n10
ad c 0 n10
ad b 10M n10
ad as 1 n10
```

We get:

```
dc=0.0002158906915
c=2.929386532e-17
b=0.01038967496
as=0.9975015621
```

Thus the carrier power (`c`) is approximately zero, the signal sideband amplitude around 1 and the DC power is given by $2b^2$.

Using the Schottky formula, we expect to have a shot noise spectral density *before the demodulation* of

$$\begin{aligned}\sqrt{S_P(f)} &= \sqrt{2\tfrac{2\pi\hbar c}{\lambda}2b^2}/\sqrt{\mathrm{Hz}} \\ &= 8.962\,10^{-12}\ \mathrm{W}/\sqrt{\mathrm{Hz}}.\end{aligned} \tag{6.26}$$

The `shot` detector in FINESSE gives `8.978e-12`. The transfer function then computes as 0.041454868 which is exactly $4ba_s$. Please note that in the presence of modulation sidebands this is not quite correct. However, in many cases it can be used as a good approximation.

Signal-to-noise

The transfer function for an optical path length difference is once more obtained by multiplying the above result by $2\pi/\lambda$ and dividing by a factor of 4:

$$T_{\Delta L'\to P} = \frac{2\pi}{\lambda} b\sqrt{P_0}. \tag{6.27}$$

Now we have to propagate the shot-noise through the demodulator as well. Please consider that this is *not* done automatically by FINESSE even if you use a detector like, for example, `pdS2`. We consider the amplitude spectral density $\sqrt{S_P(f)}$: because we are only interested in the DC signal after the demodulation we can approximate the white spectrum by two uncorrelated noise amplitudes at $\pm\omega_m$ with ω_m being the demodulation frequency. Thus the amplitude noise spectral density after demodulation can be approximated as:

$$\left[\sqrt{S_P(f)}\cos(\omega_m t)\right]_{|DC} \approx \left(A_1\cos\left(-\omega_m t\right) + A_2\cos\left(\omega_m t\right)\right)\cos(\omega_m t) \tag{6.28}$$

with A_1 and A_2 as two uncorrelated amplitudes[1] of the magnitude $\sqrt{S_P(f)}$. The right side of above equation yields:

$$\frac{1}{2}(A_1 + A_2) = \frac{1}{\sqrt{2}}A_1 = \frac{1}{\sqrt{2}}\sqrt{S_P(f)} = \sqrt{\frac{2\pi\hbar c}{\lambda}2b^2}/\sqrt{\mathrm{Hz}} \tag{6.29}$$

[1] It should be clear that in a realistic scenario the noise amplitudes cannot be assumed to be always uncorrelated.

This yields an apparent mirror motion of:

$$\begin{aligned}\sqrt{S_{\Delta L}(f)} &= \frac{\sqrt{\frac{2\pi\hbar c}{\lambda}2b^2}}{\frac{2\pi}{\lambda}b\sqrt{P_0}} = \sqrt{\frac{\hbar c\lambda}{\pi P_0}} \\ &= 1.0341010^{-16}\,\mathrm{m}/\sqrt{\mathrm{Hz}},\end{aligned} \tag{6.30}$$

which is the same as in the case of the Michelson interferometer at a half fringe with two detectors.

Chapter 7

Advanced Usage

This chapter collects some thoughts and examples which might be of interest to more advanced users.

7.1 PyKat: Finesse and Python

During the past years our group has used MATLAB as a standard software for tasks such as designing experiments, analysing data and especially in connection with Finesse, see section 7.2 below.

Meanwhile the programming language Python has been extended by powerful modules which can replace most of the basic MATLAB functionality. Python and the modules for scientific computations are free and open source while MATLAB licenses can be expensive. Furthermore some aspects of the MATLAB language are not ideally suited for some tasks, typically handling text based files, which is sometimes of interest, for example, for the automation of complex simulations. For several years we had planned to start a new project, creating a Python tool set for the use with Finesse. This project [PyKat] is under development now.

PyKat is a free Python interface and set of tools to run Finesse. It has been used by us to perform and document a series of simulations started during the Commissioning Workshop at the LIGO detector in Louisiana in 2013. The results are collected in the document 'Comparing Finesse simulations, analytical solutions and OSCAR simulations of Fabry-Perot alignment signals', freely available online: http://arxiv.org/abs/1401.5727. The PyKat and Finesse files used to perform the simulations reported are included as an example in the PyKat package (pykat/examples/asc_test).

If you want to try out PyKat, see http://www.gwoptics.org/pykat/. The project is still in an early stage and very actively developed at the moment. We welcome all kinds of ideas, suggestions and contributions.

7.2 FINESSE and MATLAB (Octave[1])

MATLAB has become a quasi standard tool for solving numerical analyses in many areas of science, for example, the interferometer design and commissioning of gravitational wave detectors utilises MATLAB in various ways. For convenience and consistency it is helpful to provide interfaces between FINESSE and MATLAB.

There are the following three main ways to use FINESSE with MATLAB (or Octave):

- plotting via the automatically generated MATLAB function: running a FINESSE simulation provides a number of output files, one of which is a MATLAB *.m file containing a function. This function can be called from MATLAB to automatically plot and/or load the simulation output. See Section 2.6 for more details on the usage of these files.
- running FINESSE simulations from the MATLAB command window: A set of MATLAB functions, called *SimTools* is available from the FINESSE download page. The functions should enable you to read, change and write FINESSE input files from within MATLAB, as well as to start a simulation and read in the output data, see section 7.2.1.
- Communicating directly with a running FINESSE process from within MATLAB: FINESSE can be used in a client-server mode, in which a MATLAB client can talk via an internet (TCP/IP) connection to the FINESSE process, see section 7.2.2. This is the most powerful method for using FINESSE with MATLAB as it is not restricted to the usual 'xaxis' tuning style but can be used for entirely different types of simulation tasks, such as *tolerancing* which is part of many commercial packages.

7.2.1 SimTools

Triggered by similar work by Seiji Kawamura and Osamu Miyakawa, I started to use Octave to post-process the output data of FINESSE. In the course of the Virgo commissioning activity, Gabriele Vajente then developed a set of simple scripts that automate certain computation tasks nicely. During the design process for second generation gravitational wave detectors the simulation tasks became more complex still and I needed to improve the automation of tasks further. Therefore, I have started to provide a consistent set of MATLAB functions, called *SimTools* [SimTools], that can be used to read, write, edit and execute FINESSE input files. By now the SimTools includes a great number of utility functions to read, write and parse any text based simulation input file but also optics function, from FFT propagation to ABCD matrix computations[2]. We have made

[1] Octave is a GNU software package similar to MATLAB. The examples shown here can often be used with both programs, maybe after some small changes. I have not tested any of the files for such compatibility though.

[2] It should be notes that the SimTools package is far from an elegant solution, it has grown historically from a set of utility scripts and in particular the parsing of text is a workaround. It has proven to be very useful but is difficult to maintain. Eventually I would like to rewrite this as a Python package,

extensive use of SimTools for the processing and use of mirror surface maps.

The basic idea behind SimTools is to separate FINESSE input files logically into smaller parts which can be handled separately. Many of the SimTools functions deal with reading and parsing of FINESSE input files. The two main elements of the text parsing are *text lines* and *text blocks*. The latter are identified by a special comment in the input file, for example, the following creates a block with the name 'cavity':

```
%%% FTblock cavity
m m1 0.9 0.1 0 n1 n2              # mirror m1 with R=0.9, T=0.1, phi=0
s s1 1200 n2 n3                   # space  s1 with L=1200
m m2 0.8 0.2 0 n3 n4              # mirror m2 with R=0.8, T=0.2, phi=0
%%% FTend
```

The SimTools function can be used to recognize, read and edit such blocks. The following example code should give you a first idea on how this can be used:

```
% name of kat file which contains 'blocks'
inname='testconsts.kat';

% read in block from testblock.kat
block=FT_read_blocks_from_file(inname);
myblock=FT_copy_block(block,'constants');

% print reflectivies
r1=FT_read_constant(myblock,'Rm1');
r2=FT_read_constant(myblock,'Rm2');
disp(sprintf('Reflectivities of m1 and m2: %f %f',r1,r2));

% now we change the reflectivity for one of m
myblock=FT_write_constant(myblock,'Rm1',0.7);
```

SimTools are developed independently from FINESSE itself and therefore will not be described in detail in this manual. Please download the SimTools package from the FINESSE download page for more information.

7.2.2 Client-Server mode of FINESSE

The Linux and Mac OS X versions of the FINESSE binary can be started in a so-called *servermode* by calling the program with:

```
kat --server <portnumber (11000 to 11010)> [options] inputfile
```

The port number can be chosen by the user, the other options may be any of the usual options for calling FINESSE. Also, the input file can be any unchanged input file. For example, we might load the file `bessel.kat` with

```
kat --server 11000 bessel.kat
```

using a more consistent approach.

The input file will then be read and pre-processed as usual but instead of actually performing the simulation task (i.e. running along the xaxis) FINESSE will become idle and listen to incoming TCP/IP connection via the user-defined port (11000 in this example).

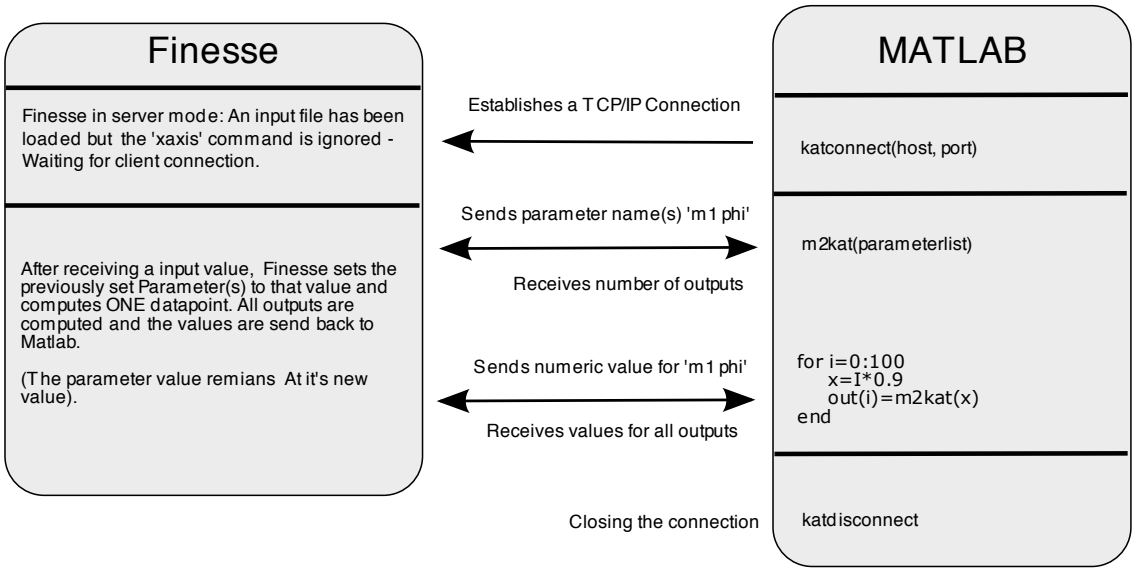


Figure 7.1: *This sketch illustrates how the communication between Finesse and MATLAB/Octave would work in a simple example.* FINESSE *loads an input file and then becomes idle until a 'katconnect' command from a MATLAB session opens a TCP/IP connection. Then the 'm2kat' command can be used to send or receive data over the connection. Typically 'm2kat' is first used to specify which data is transferred (CONFIG) and then is used again to send and receive numerical data (TUNE). After the simulation task, the command 'katdisconnect' must be used to close the TCP/IP connection again.*

A MATLAB/Octave client can then send commands via TCP/IP to FINESSE, see Figure 7.1. This works by three new MATLAB functions (from source files which need to be compiled first, see below):

- katconnect: establishes a connection with the FINESSE server
  `usage: socket=katconnect('server', port (11000-11010))`
  where 'server' is the network address of the server (use 'localhost' if you do all this locally on the same computer) and 'port' the port number chosen when starting the FINESSE server. 'socket' will return the index of the opened socket.
- katdisconnect: closes a connection with the FINESSE server
  `usage: katdisconnect(socket)`
  with 'socket' the socket number received with the 'katconnect' command.
- m2kat: performs all communications through the TCP/IP connection This commands can
  - set a certain parameter to a new numeric value
  - receive the value of a parameter

– receive output data, for example, the photodiode outputs.

'm2kat' has three different usage modes, these are chosen automatically depending on the specified input and output arguments:

```
usage:
  CONFIG:
  [number_of_outputs]=m2kat(socket, number_of_parameters, 'parameter string')
  send parameter names to be tuned and get number of output data values
or
  TUNE:
  [output_data] = m2kat(socket, number_of_outputs, parameter_values)
  send parameter values and get output data
or
  INFO:
  [output_data] = m2kat(socket, number_of_parameters)
  get current values of parameters defined by a previous call of
  m2kat
```

The CONFIG mode is required in advance of TUNE or INFO commands. The CONFIG command tells the server which parameters of the interferometer will be set or polled in the following session. With a TUNE command one can set new numerical values to these parameters, whereas an INFO command would return their current numerical values. A TUNE command would also return one output data point, i.e. one numerical value for each output specified in the input file. The usage of 'm2kat' is a bit complex and sensitive to mistakes. It therefore requires a careful preparation of the MATLAB client script. Please look at the provided example for further guidance.

Example MATLAB client file This example recreates a normal FINESSE simulation by tuning one parameter and printing the output.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% m2katexample.m   MATLAB Script for testing the MATLAB-to-Finesse
% functions katconnect, m2kat and katdisconnect
%
% Version 1.0
%
% Andreas Freise adf@star.sr.bham.ac.uk
% 29.09.2006
%
%----------------------------------------------------------------
%
% The socket connection does not do any error checking
% in order to get maximum speed. Therefore you have to
% follow precisely the correct procedure outlined below:
%
% 0. start a Finesse server (somewhere). For example,
%    on you linux computer cp1.aei.mpg.de you could do:
```

```
%      kat --server 11000 cavity1.kat
%    which reads and initialises the simulation specified in cavity1.kat
%    and then waits for a TCP/IP connection on port 11000
%
% 1. open a socket with
%    'socket=katconnect(servername, portnumber)'
%    with
%    - 'servername' being a full DNS name, e.g. cp1.aei.mpg.de, of the
%      computer which runs the Finesse server
%    - 'portnumber' a number between 11000 and 110010 (the
%      same portnumber used by the server)
%
% 2. define a list of parameters to be tuned as a string.
%    The string must consist of pairs of names:
%    parameterlist='componentname1 parametername1 componentname2 parametername2 ...'
%    e.g. parameterlist='mirror1 phi'
%
% 3. Send a 'CONFIG' command to the Finesse server with:
%      nout=m2kat(socket,noparams,parameterlist)
%    with
%    - 'socket' the socket number
%    - 'noparams' the number of parameters (number or word pars in 'parameterlist')
%    - 'parameterlist' the string described above
%    - 'nout' the number of values that the server will return for
%      each computation
%
%    Now the server ready and waits for numeric input.
%
% 4. Send 'TUNE' commands to the server. A TUNE command is followed
%    by a number of input values and returns computation results
%    from the server:
%      data=m2kat(socket,nout,[inputvalues]);
%    with
%    - 'socket' the socket number
%    - 'nout' the number of expected values being returned from the
%      server
%    - '[inputvalues]' the parameter values, i.e.a vector of double
%      values, representing the new value the respective parameter
%      shall be tuned to. The first number refers to the first
%      parameter in 'aprameterlist', etc.
%    - 'data' being the result, a vector of doubles (the size of the
%      vector is given in 'nout' and is determined by the number of
%      detectors or other outputs specified in the Finesse input
%      file, e.g. cavity1.kat
%
% 5. Now you can send as many TUNE commands as you like by calling
%    repeatingly 'm2kat'.
%
% 6. You can send a CONFIG command again, if you like to start
%    tuning different parameters at some point
```

```
%
% 7. When you have finished, it's important to close the socket
%    with
%      katdisconnect(socket)
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% server name and portnumber
hostname='localhost';
port=11000;

% parameter(s) to be tuned
noparams=2; % number of parameters
parameterlist='m1 phi m2 phi'; % list of parameters

% defining the x-axis
N=20001;
min=-10;
max=180;
x=linspace(min,max,N);

%%%%%%%%%%%%%%%%%%%%%%%%
% Variables for progress display
max_calls=N;
curr_call=0;
call_range=round(N/100.0)+1;
last_print=0;
%%%%%%%%%%%%%%%%%%%%%%%%
tic

% opening socket
socket=katconnect(hostname,port);

if (socket>0)
  % send CONFIG command and parameter list
  nout=m2kat(socket,noparams,parameterlist);
  if (nout>0)
    % prepare matrix for output data
    out=zeros(N,nout);
    % run along xaxis
    disp(sprintf(' \n'));
    disp(sprintf(' \n'));
    for i=1:N
      % compute one data point
      out(i,:)=m2kat(socket,nout,[x(i),0.0]);
      % compute and print progress
      curr_call=curr_call+1;
      last_print=last_print+1;
      if (last_print>=call_range)
```

```
        disp(sprintf('\b\b\b\b\b\b\n%3d%%',round(100*curr_call/ ...
                                              max_calls)));
        last_print=0;
      end
    end
  end
end

disp(sprintf('\b\b\b\b\b\b\n100%% -- Complete!\n'));
toc
% send QUIT command and close socket
katdisconnect(socket);
% plot result
plot(x,out);
```

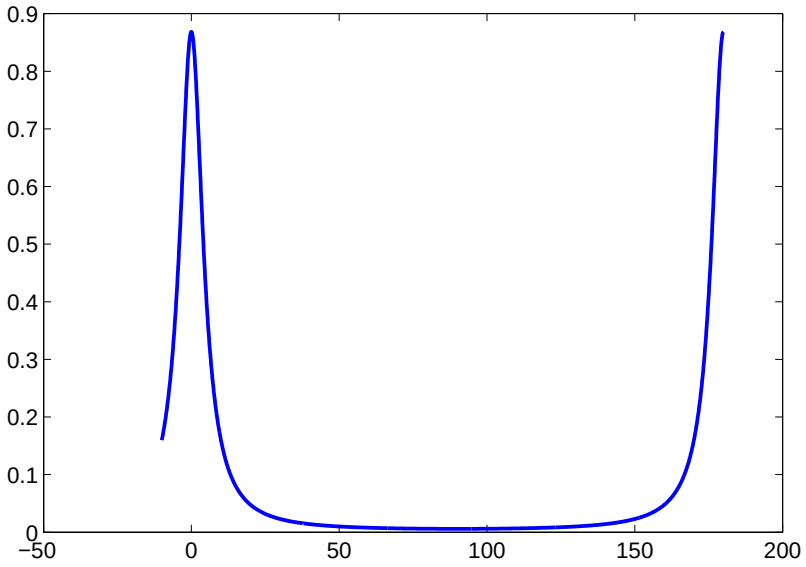


Figure 7.2: *Graphical output of the example script demonstrating the communication between MATLAB and* FINESSE *running in server mode.*

The terminal output of FINESSE during the example above would look similar to this:

```
moon:~/work/kat/test/m2kat$ kat --server 11000 cavity1.

------------------------------------------------------------------------
                    FINESSE 0.99.7 (build 3227)
      o_.-=.        Frequency domain INterferomEter Simulation SoftwarE
     (\'".\|        03.07.2008      A. Freise  (afreise@googlemail.com)
     .>' (_--.
   _=/d   ,^\       Input file cavity1.kat,
  ~~ \)-'   '       Output file cavity1.out,
     / |            Gnuplot file cavity1.gnu
    '  '                                    Wed Jul  9 11:33:04 2008
------------------------------------------------------------------------
```

```
*** cmd_process: processing [0]...done.
*** listen: listening on port 11000
*** cmd_process: processing [0]...done.
*** cmd_process: processing [0]...done.
*** cmd_process: processing [0]...done.

*** listen: accepted a connection from 127.0.0.1:-2491
*** listen: num connections = 1

*** receivecommands: processing CONFIG command
Number of Parameters = 2
Read parameter list: m1 phi m2 phi
*** receivecommands: received 2 parameters
*** receivecommands: processing quit command
*** listen: num connections = 0
*** cmd_process: processing [0]...done.
```

The first line starts FINESSE in server mode, the banner is printed and FINESSE starts to ''listen' for incoming TCP/IP connections. It then accepts a connection from IP number 127.0.0.1 (this means the MATLAB client I am using runs on the same computer). The following lines acknowledge the receipt of a 'CONFIG' command. The following 'TUNE' commands do not produce any terminal output. During this example run MATLAB would produce terminal output as follows:

```
>> m2katexample
*** Creating Socket ...
*** server name is localhost
*** server internet name is localhost
*** connecting to localhost
*** Socket 9 opened!
*** CONFIG: OK

100% -- Complete!

Elapsed time is 0.449107 seconds.

### Closing socket 9 ...   ... Done!
>>
```

The graphical output of course depends on the details of the finesse input file. Figure 7.2 shows the output of this example.

Compiling the Client programs

The programs are scripts that run in MATLAB and Octave. For simplicity I do not provide binary versions of these, however, they are very easy to compile.

Compiling in MATLAB The MATLAB client consists of a number of C source files which have to be compiled with the MATLAB compiler. The files are:

```
katconnect.c      : create TCP/IP connection with server
katdisconnect.c   : break TCP/IP connection
m2kat.c           : send commands to and receive data from server
m2kat.h           : include file for m2kat.c
```

These files can be compiled from within MATLAB with the mex command. For example, on a Macintosh the command (used inside the MATLAB command window)

```
mex katconnect.c
```

will create the binary file 'katconnect.mexmaci'. Once this file exists you can call `katconnect` or do `katconnect` just like with any MATLAB command. Please compile the three files 'katconnect', 'katdisconnect' and 'm2kat' and keep all files, source and binary, in a directory where MATLAB can find them.

Compiling in Octave The compilation from Octave works exactly as above. You can use the 'mex' command from Octave with the same source files. The only difference is that that the mex command in Octave will create '*.o' and '*.mex' files storing the binary commands.

Appendix A

Tutorial for setting up and locking a cavity

This sections is an attempt to records a step-by-step description of how to develop a FINESSE input file for a cavity lock from scratch. The tutorial has been given during the meeting of the GEO 600 simulation group on the 13.03.2007 in Hannover.

A.1 The Basics

We assume that we know some data about the cavity from previous calculations and create the respective input file:

- laser: $\lambda = 1064\,\mathrm{nm}$, $P = 1\,\mathrm{W}$, $w_0 \approx 1\,\mathrm{mm}$
- cavity input mirror: flat, $R = 0.9$
- cavity end mirror: $R_C = 1\,\mathrm{m}$, $R \approx 1$
- cavity length: $L = 0.8\,\mathrm{m}$

```
l laser 1 0 n1
gauss beam1 laser n1 1m 0

s s1 1 n1 n2

m m1 0.9 0.1 0 n2 n3
s sL 0.8 n3 n4
m m2 1 0 0 n4 dump

maxtem 0
pd cavpower n4
xaxis m1 phi lin -10 10 300
```

This basic syntax can be found in the example files that come with FINESSE. In addition you can always use the help page with `kat -h`.

Now for the curvature of the end mirror. It is always tricky to understand how the sign of the curvature is defined. We type `kat -hh` to get some information from the second help page (there are exactly two):

```
** Geometrical conventions:
 tangential plane: x, z (index n), saggital plane: y, z (index m)
```

```
xbeta refers to a rotation in the x, z plane, i.e. around the y-axis
R<0 when the center of the respective sphere is down beam
(mirror: node1 -> node2, beam splitter: node1 -> node3)
beam parameter z<0 when waist position is down beam
```

Thus in this case the radius of curvature for `m2` is defined positive and we add `attr m2 Rc 1`. Please note that this definition depends directly on the order in which the node names are given. For example, if we change the line for the end mirror to

```
m m2 1 0 0 dump n4
```

the curvature would have to be set as $-1\,\mathrm{m}$.

In Gerhard Heinzel's thesis we can find many useful equations for two mirror cavities. From these we expect the circulation light power to be $\approx 4/T = 40\,\mathrm{W}$. But running the current file we obtain only $P < 0.1\,\mathrm{W}$. Something is wrong. Maybe we must we set `maxtem` higher? Trying that we obtain:

maxtem	0	2	4	12
power [W]	0.01	0.04	0.12	2.6

So, what's wrong? We are not mode-matched, OK, but it seems like we are not converging with higher mode numbers either. The answer is that we are not using the right base set for the TEM modes. In that case not only the amplitudes but also the phases of the higher order modes play important roles in creating the cavity resonance. Very many higher order modes are required to obtain correct results for a cavity without an optimised set of base modes. **If all possible you should use the cavity eigenmodes.**

In order to switch the beam parameters to the cavity eigenmodes we can use the `cav` command:

```
cav cav1 m1 n3 m2 n4
```

This will automatically compute the cavity eigenmodes and use those as for computing the cavity fields.

With the `cav` command we obtain a power of 14.66 W independent of the settings for `maxtem`. Looks much better. How can we now modematch the laser beam to the cavity? The most simple way is to omit the gauss command. This will tell FINESSE to trace the cavity mode back to the laser and assume that as the correct mode for the input beam. With that we get a power of 37.97 W, again independent of the `maxtem` value. That is good enough for a first check (the 40 W computed above where an approximation).

No we can ask ourselves what would have been the correct beam parameter at the laser for a modematched beam? We can find out using the `trace` command. The help pages (`kat -hh`) tell us:

```
** trace n: 'n' bit coded word, the bits give the following output:
    trace 1:   list of TEM modes used
    trace 2:   cavity eigenvalues and cavity parameters like FWHM,
               FSR optical length and finesse
    trace 4:   mode mismatch parameters for the initial setup
```

```
trace 8:   beam parameters for every node, nodes are listed in
           the order found by the tracing algorithm
trace 16:  Gouy phases for all spaces
trace 32:  coupling coefficients for all components
trace 64:  mode matching parameters during calculation, if they
           change due to a parameter change, for example by
           changing a radius of curvature.
trace 128: nodes found during the cavity tracing
```

We run the file again, adding the command `trace 8` to the file and obtain the following text output:

```
--- tracing the beam through optical system,
 found 4 of 5 nodes:
 (w0 : waist radius, w : beam radius, z: distance to waist,
  z_R: Rayleigh range, q : complex beam parameter
  and gamma: far field angle
0: node n3(2); m1(0), sL(3); n=1  (m1 --> n3)
   x, y: w0=368.06615um w=368.06615um z=0m z_R=400mm
   q=0.4i gamma=920.16536urad
1: node n4(3); sL(3), m2(1); n=1  (sL --> n4)
   x, y: w0=368.06615um w=823.02092um z=800mm z_R=400mm
   q=(0.8 + 0.4i) gamma=920.16536urad
2: node n2(1); m1(0), s1(2); n=1  (m1 --> n2)
   x, y: w0=368.06615um w=368.06615um z=0m z_R=400mm
   q=0.4i gamma=920.16536urad
3: node n1(0); s1(2), laser(4); n=1  (s1 --> n1)
   x, y: w0=368.06615um w=991.04843um z=1m z_R=400mm
   q=(1 + 0.4i) gamma=920.16536urad
```

This shows the beam parameters at every node. Please remember that the sign of z depends on the direction indicated by the arrow: component *rightarrow* component.

We find the right beam parameter at the node `n1` to be: w_0 =0.4 mm and $z_0 = -1$ m. We test this using `gauss beam1 laser n1 .4m -1` and `maxtem 0`. In this case the cavity eigenmodes are used within the cavity and the explicitly given laser mode outside the cavity. Again we get a power of 37.7 W, we are mode-matched.

A.2 Creating the error signal

The FSR of the cavity is given as $c/2L = 187.5$ MHz. FWHM should be about 3 MHz (as the finesse is $2\pi/T \approx 60$).

We chose a modulation at 30 MHz and change the file to

```
s s1 .5 n1 n1a
mod eom1 30M 0.3 1 pm n1a n1b
s s2 .5 n1b n2
```

We want to detect the signal in reflection. It is useful to remember that the following simulation leads to a very boring plot:

```
pd refpower n2
xaxis m1 phi lin -90 90 300
```

BTW the result is not equal to 1 since we don't include all the sidebands created by the modulator. You can try to use the modulator with more higher orders:

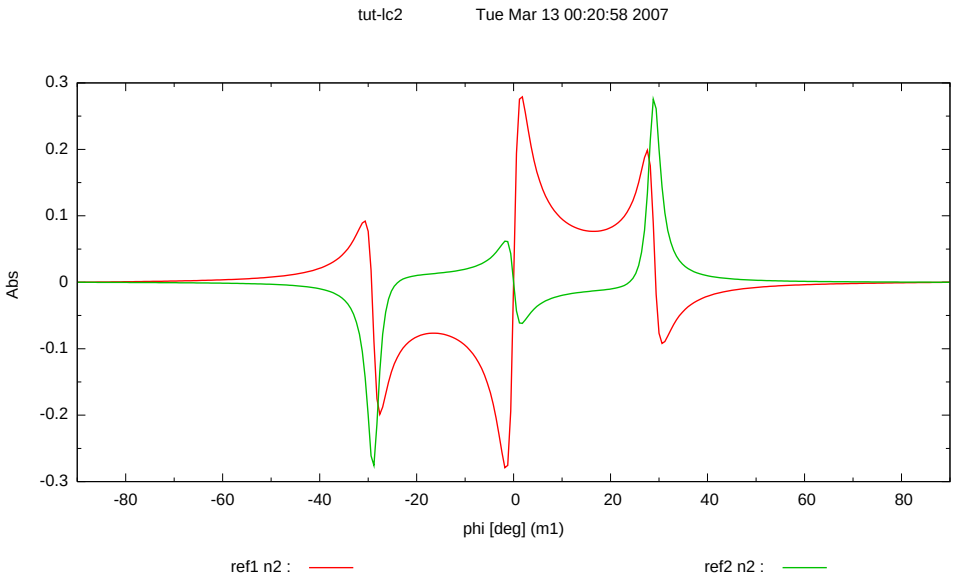
```
mod eom1 30M 0.3 5 pm n1a n1b
```

and you'll recover the lost laser power.

To see an error signal in reflection we need to demodulate the photo-current. We try rather arbitrarily demodulation phases 0 and 90 deg:

```
pd1 ref1 30M 0 n2
pd1 ref2 30M 90 n2
```

and obtain:



which looks like a proper PDH signal.

Further optimization works better using a transfer function. To create a transfer function we need to add a signal (at the mirror we have been tuned before). Furthermore we need to use a `pd2` detector and sweep the frequency with the `xaxis` command

```
fsig sig1 m2 100 0
pd2 ref2 30M 0 1 n2
xaxis sig1 f log 10k 10M 300
put ref2 f2 $x1
```

The put command is required to sweep the demodulation frequency of the phot detector in sync with the signal frequencies (all parameters in FINESSE are independent of each other. You need to use `put` and maybe `func` commands to link them if necessary).

This yields the typical low-pass transfer function of a cavity. The DC limit represents the *optical gain* of the cavity system. We can optimise the gain by tuning the demodultion phase (of the modulation frequency). To do se we chose a low frequency for the signal (1 Hz as we already did). Then we use the `xaxis` command to tune the demodulation phase:

```
xaxis ref2 phi1 lin 0 180 300
```

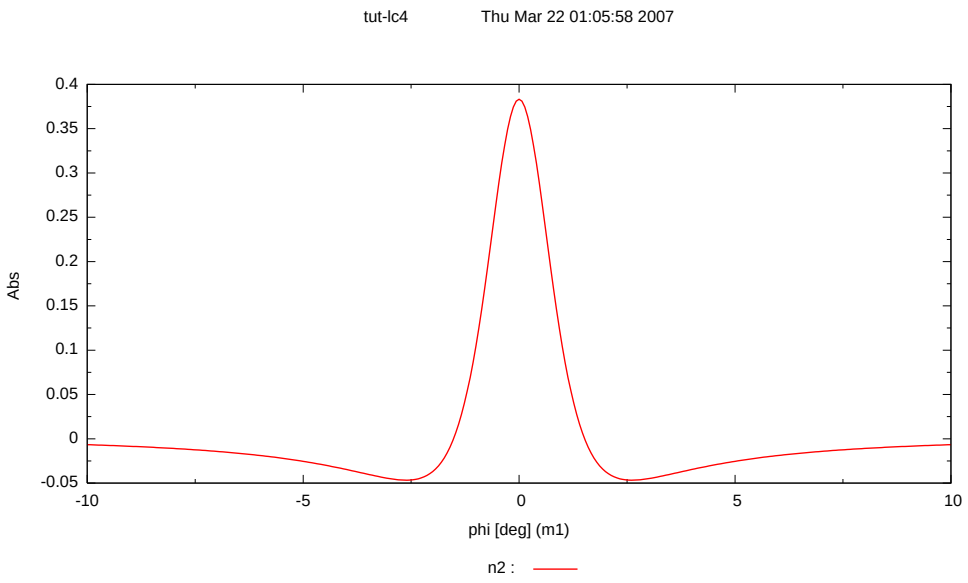
The maximum signal corresponds to a demodulation phase of 167 deg.

We can also see that the optical gain is equal to ≈ 22 and the unit is [W/rad].

For the `lock` command (see below) we need to know the optical gain in [W/deg]. The conversion factor can be computed as follows: 360 degrees correspond to a movement of one wavelength. Equally 2π [rad] refer to one wavelength. Thus $1\,\mathrm{W/rad} = 2\pi/360\,\mathrm{W/deg}$ and we obtain an optical gain of $\approx 0.4\,\mathrm{W/deg}$. To test this we can go back to a `pd1` photodetector (now using the optimised demodulation phase of 167 deg). Again we move the tuning of `m2` with the `xaxis`. To compute the slope we can use the `diff` which can performa differentiation:

```
diff m2 phi
```

and obtain the gradient of the error signal. Since the tuning with xaxis is in [deg] the resulting units are [W/deg]. The slope in the oprating point (phi=0) is about 0.4 as expected.



A.3 Forming the locking loop

The photodiode above crates the required error signal. The `lock` command can be used to perform an iterative loop using such an error signal. To do so we first need to access

the signal through a variable. We use the `set` command to do so:

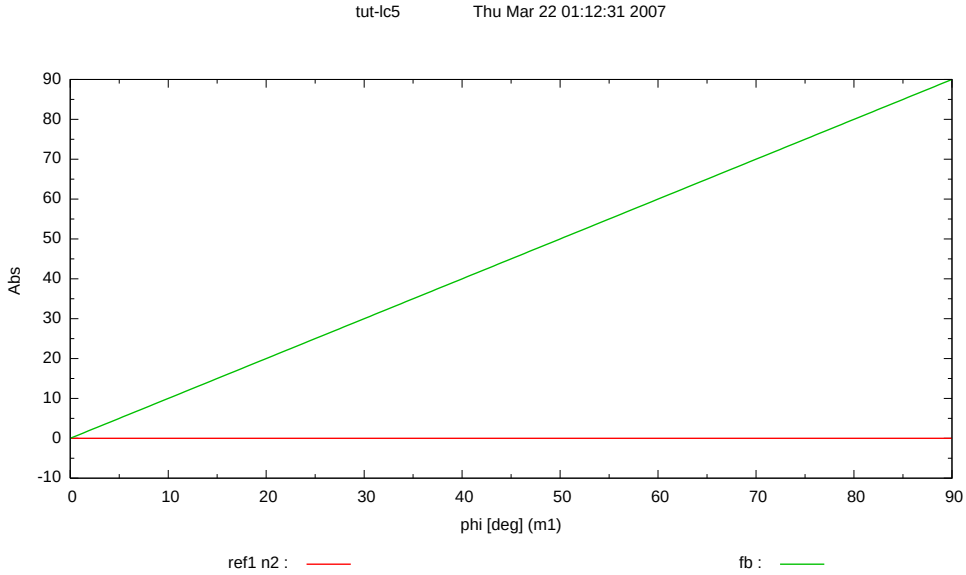
```
pd1 ref1 30M 167 n2
set err ref1 re
```

This defines a new parameter `err` which is linked to the real part of the output signal of photodetector `ref1`. Thus we have the (optimised) error signal available in `err` Next we can create the feedback signal and connect it to the system. The feedback signal is created by the `lock` command and then can be fed to the position (tuning) of `m2` with a `put` command:

```
lock fb $err -2 1m
put m2 phi $fb
```

The `lock` command required two numeric parameters: the gain and accuracy of the loop (please look up the syntax of the `lock` command in the manual). The gain in the `lock` command should be $-1/(\text{opticalgain})$. Thus we set it to $-1/0.4 \approx -2$. We will discuss the accuracy below and arbitrarily set it to 1/1000.

To test this we move `m1`:



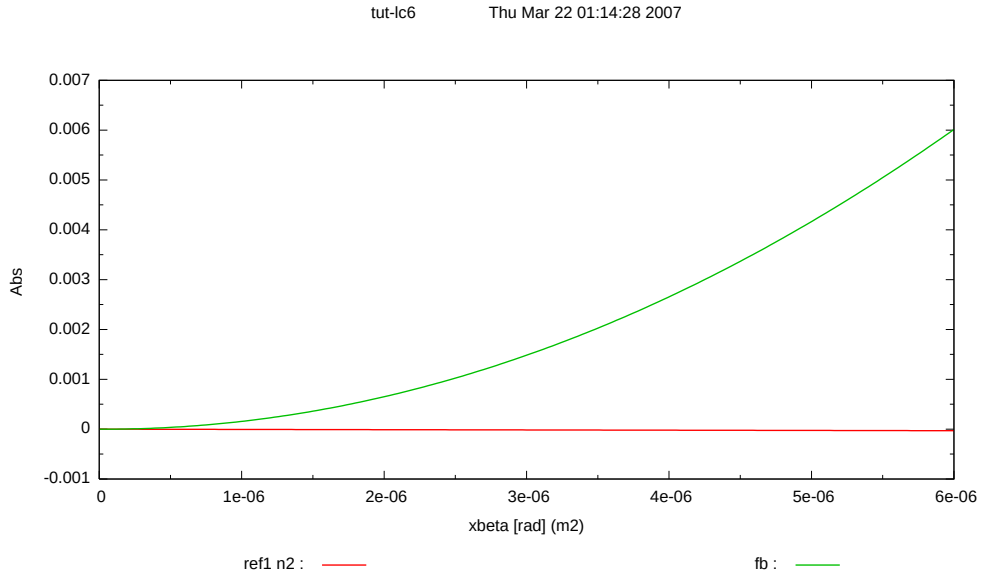
We see that m2 follows the tuning of m1 and the error signal held below 1 mW. The accuracy refers to error signal but with the optical gain we can also compute the residual noise in terms of mirror motion. We know that 1 W refers to 1/22rad and 1 rad refers to a motion of $\lambda/(2\pi)$. Thus we get an accuracy of 45 μrad or 7 pm.

It is important to start all tunings in or close to the operating point (all tunings equal to zero). Also the step size must be small enough the remain always close to the operting point. Otherwise the lock will fail.

Another test: we can misalign m2:

```
xaxis m2 xbeta lin 0n 6u 200
```

(Do not forget to use at least `maxtem 1` with this). This yields:



And this is the file for the locked cavity with the commands set for the last test above:

```
l laser 1 0 n1

s s1 .5 n1 n1a

mod eom1 30M 0.3 1 pm n1a n1b

s s2 .5 n1b n2

m m1 0.9 0.1 0 n2 n3
s sL 0.8 n3 n4
m m2 1 0 0 n4 dump
attr m2 Rc 1

maxtem 1

pd1 ref1 30M 167 n2

set err ref1 re

lock fb $err -2 1m
put m2 phi $fb

xaxis m2 xbeta lin 0 6u 200

cav cav1 m1 n3 m2 n4

gnuterm pdf
```

Appendix B

Shot-noise limited sensitivity of GEO 600

This section records FINESSE simulations from 2007 which were undertaken to investigate the shot-noise limited sensitivity of the GEO 600 detector. As it turns out the calculation of the sensitivity contains a fair number of factors of 2 and $\sqrt{2}$, leading to potential confusion when comparing results. Also the GEO detector using a heterodyne read-out scheme at that time required a slightly more complicated shot-noise computation than that given in the usual textbooks. As a result our models of the GEO 600 shot-noise did not match the measured sensitivity exactly. This appendix and Section 6.2 report on the effort to validate the shot-noise model used by FINESSE and to demonstrate how to correctly derive the shot-noise limited sensitivity. The plots in Figure B.4 show that the here derived shot-noise model correctly matches the measured data.

Note that this chapter contains old results which are not entirely correct! During the implementation of the new `qshot` and `qnoised` detectors in 2013 we found that the comparison below contains one error: the correction factor of the shot noise for the presence of RF sidebands in this case varies between $\sqrt{3/2}$ and $\sqrt{2}$ depending on the demodulation phase [Meers, Niebauer]. However in the comparison below only $\sqrt{2}$ is used regardless of the phase. Thus the sensitivity plots shown in figure B.1 are a factor of $\sqrt{3}$ worse than they should be. This went unnoticed because of a bug with the original `qshot` detector. Running these example files with FINESSE 2.0, using the new `qshot` and `qnoised` detectors, will provide the correct result. Note that comparison with the measured data did not show this problem because the wrong factor affected mostly the low-frequency part and not the high-frequency part where GEO's sensitivity is shot-noise limited. **We have decided to keep the this chapter unchanged in order to keep a record of past results and of the bug in the `qshot` detector.**

B.1 The qshot command (before Finesse 2.0)

Early versions of FINESSE were able to compute only a very approximative shot noise level using the Schottky equation. The current version of FINESSE features a new command `qshot` which provides a more accurate approximation of the shotnoise in a certain photo diode signal. The `qshot` command can correctly compute the shotnoise level in the presence of a number of modulation sidebands and in the absence of squeezing and

radiation pressure effects. Details of the implemented algorithm have been collected in a paper [Harms].

B.2 Comparing the different methods

The following shows how to use the FINESSE syntax correctly to compute the shot-noise limited sensitivity of GEO 600; it explains step by step how the apparent strain is computed from the transfer function end-mirror motion $\longrightarrow$ detector output (T) and the amplitude spectral density $\sqrt{S_P}$ associated with shot-noise[1]. Six extracts from FINESSE input files will be shown below, each demonstrating a correct but slightly different method to obtain a shot-noise limited sensitivity. The first method is as follows:

```
# method 1
# adding differential displacement to the end mirrors
fsig sig1 MCN 1 0
fsig sig2 MCE 1 180

# compute Schottky noise
shot noise nMSR2
noplot noise
set sn1 noise re
# now we apply correction factors for a) the mixer and b) possible noise
# correlations the latter is between sqrt(1.5) and sqrt(2)
func sn2 = $sn1 / sqrt(2) * sqrt(2) # or sqrt(1.5) see labbook page 3911
noplot sn2

# compute transfer function  Delta_L -> DF
pd2 pdMI $fMI max 1 nMSR2
noplot pdMI
set tr1 pdMI abs
# note until the newsest finesse version 040207 'abs' actually above
# produces abs^2 !!
#convert transfer function from W/rad into W/m
func tr2 = 2* $pi / 1064.0E-9 * $tr1
noplot tr2

# compute apparent strain as h=2 Delta_L/L and
# apparent_Delta_L = shotnoise/transfer_function
func h = 2 /1200 * $sn2 / ( $tr2 + 1E-12 )

xaxis sig1 f log 10 10k 300
put pdMI f2 $x1
```

[1] Note that GEO 600 has folded arms with the folding mirrors called 'far' mirrors. The displacement should be injected at the 'central' end mirrors. If the 'far' mirrors are used the effect is amplified by a factor two due to the double reflection which must be corrected manually.

The method above is the most explicit and relies on several manual conversions and corrections which are apllied via `func` commands. The correction and conversion factors are:

a) the shotnoise is first divided by $\sqrt{2}$ to simulate the effect of the mixer, see Section 3.4.6.
b) then the shotnoise is multiplied by $\sqrt{2}$. This is a upper limit for the noise increase due to the heterodyne type measurement.
c) The transfer function is computed by FINESSE as W/rad and must be converted into W/m by mulitiplication with $2\pi/\lambda$, see Section 6.2.1.
d) Shotnoise can then be converted into apparent displacement noise by dividing it by the transfer function: $\sqrt{S_{\Delta L}} = \sqrt{S_P}/T$. It is important to understand what we mean by ΔL, namely the position change of *each* end mirror. Consider again the computation of the transfer function: We inject a signal with amplitude x to both end mirrors and compute the signal amplitude on the output photodiode; i.e. if, for example, any noise source would create exactly this amplitude on the diode the apparent displacement would correspond to a motion of *each* mirror by x. Thus the transfer function refers to ΔL as the arm length change of one individual arm.
e) With this definition of ΔL we know from Martin's thesis [Hewitson04] that the apparent strain sensitivity computes as $h = 2\Delta L/1200$. In the code the last two computations have been merged into $h = 2\sqrt{S_P}/(1200\ T)$.

The following code repeats the above in a slightly more compact form:

```
# method 2
# The same as above a bit more compact:
fsig sig1 MCN 1 0
fsig sig2 MCE 1 180
pdS2 pdMI $fMI max 1 max nMSR2
noplot pdMI
set sens1 pdMI abs
func sens2 = $sens1 / 600 * 1064.0E-9 / ( 2* $pi )/ sqrt(2) * sqrt(2)
xaxis sig1 f log 10 10k 300
put pdMI f2 $x1
```

We can also apply the signal frequencies to the spaces (this simulates GW signals) This feature is less well tested than `fsig` connected to mirrors but it provides a good redundant check on the result above:

```
# method 3
fsig sig1 snorth1 1 0
fsig sig2 snorth2 1 0
fsig sig3 seast1 1 180
fsig sig4 seast2 1 180
```

```
pdS2 pdMI $fMI max 1 max nMSR2
noplot pdMI
set sens1 pdMI abs
func sens2 = $sens1 / sqrt(2) * sqrt(2)
xaxis sig1 f log 10 10k 300
put pdMI f2 $x1
```

Instead of the Schottky formula we can use the qshot detector which includes the effects of the RF modulation. Thus, we do not apply an extra the 'correction factor' for modulation sidebands ($\sqrt{2}$ in the example above) nor the $1/\sqrt{2}$ for the demodulation by the mixer:

```
# method 4
fsig sig1 MCN 1 0
fsig sig2 MCE 1 180
# compute shotnoise
qshot qnoise 2 $fMI max 1 max nMSR2
noplot qnoise
set qn1p qnoise re
# now compute the linear spectral desnity i.e.
# noise = sqrt (qshot * h * f0)
func qn2p = sqrt( $qn1p * 6.6262E-34 * 299792458.0 / 1064E-9 )
noplot qn2p
# compute transfer function  Delta_L -> DF
pd2 pdMI $fMI max 1 max nMSR2
noplot pdMI
set tr1p pdMI abs
# convert transfer function from W/rad into W/m
func tr2p = 2* $pi / 1064.0E-9 * $tr1p
noplot tr2p
# compute apparent strain as h=2 Delta_L/L and
# apparent_Delta_L = shotnoise/transfer_function
func hp = 2 /1200 * $qn2p / ( $tr2p + 1E-12 )
xaxis sig1 f log 10 10k 300
put pdMI f2 $x1
put qnoise f2 $x1
```

Or using qshotS:

```
# method 5
# The same as above a bit more compact:
fsig sig1 MCN 1 0
fsig sig2 MCE 1 180
qshotS pdMI 2 $fMI max 1 max nMSR2
set s1 pdMI abs
```

```
# compute sens in m/sqrt(Hz)
const c  299792458.0
const h 6.6262E-34
func s2 = 1064.0E-9 / 2 /$pi * $s1 * sqrt( $h * $c / 1064E-9 )
noplot s2
# compute h in 1/sqrt(Hz)
func s3 = 2/1200 * $s2
noplot pdMI
xaxis sig1 f log 10 10k 300
put pdMI f2 $x1
```

Or using displacement and qnoiseS and the scale command:

```
# method 6
fsig sig1 MCN 1 0
fsig sig2 MCE 1 180
qshotS pdMIS 2 $fMI max 1 max nMSR2
set s1 pdMIS abs
noplot pdMIS
func s3 = 2/1200 * $s1
scale qshot:meter s3
xaxis sig1 f log 10 10k 300
put pdMIS f2 $x1
```

The good news is that these methods agree with each other, see Figure B.1. Furthermore they also agree well with the measured sensitivity, see next section.

B.3 Computing the shot-noise-limited sensitivity of GEO

First we measured the light power in the south port (just after MSR) to be the same as the measured 43 mW, and tune the FINESSE file accordingly

```
#l i1 3.2 0 nMU3in1            # nominal S5 corrsponds to 75deg
l i1 3 0 nMU3in1               # tuned down from 3.2 to get right
                               # power in DF
```

The following analysis has used the 'typical S5 sensitivity' from the GEO sensitivity webpage (http://www.geo600.uni-hannover.de/geocurves/) as a reference. The respective data has been taken on 03.06.2006.

We now need to tune the demodulation phase for the P and Q channel of the dark fringe output signal. This is important to compare the simulation with measured data correctly. The optical gain is modelled for the data analysis (h reconstruction) with a simple transfer function represented by a gain, a complex pole and a real zero. The numerical values for these parameters on the 03.06.2006 for the P and Q channel respectively can be found on

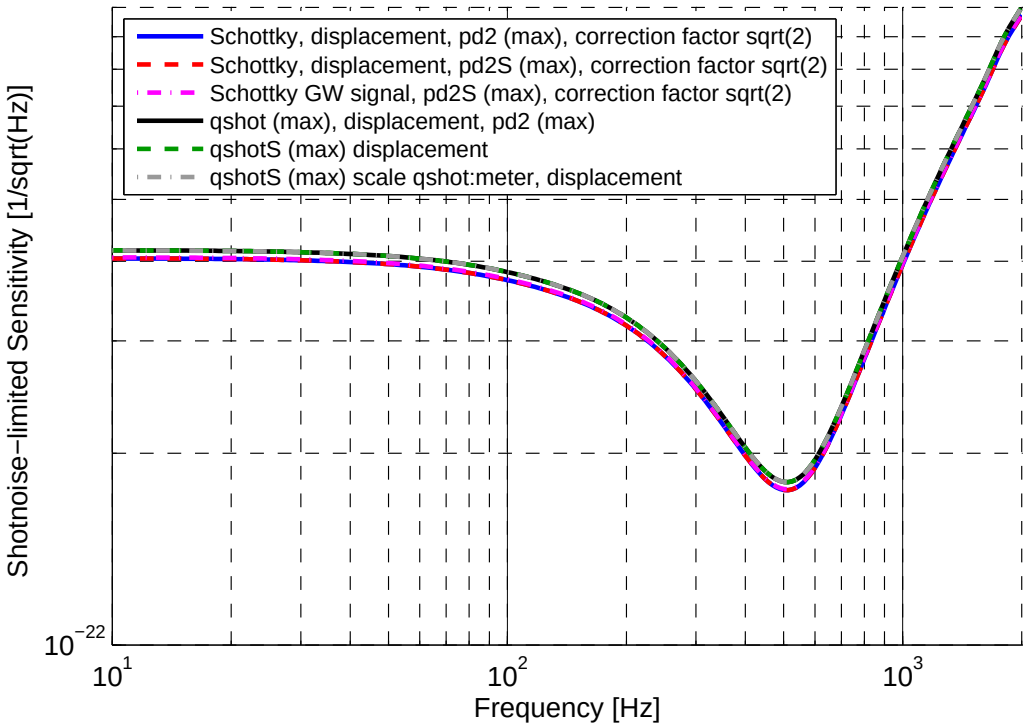


Figure B.1: ***The sensitivity plots here are a factor of $\sqrt{3}$ worse than is correct, see the beginning of this chapter for more details.*** *Shot-noise-limited sensitivity of GEO 600 as computed with* FINESSE. *The plot compares the six different methods to compute a shotnoise-limited sesnitivity mentioned in the text:* **Method 1 and 2** *use* `fsig` *to inject differential displacement noise to the near end mirrors; the difference lies only in how the conversion into apparent strain is performed.* **Method 3**, *however, uses* `fsig` *to add simulated gravitational-wave signals to the lengths between the far and central mirrors.***Method 4** *employs the new* `qshot` *detector. This detector makes use of a new algorithm to take into account the effects of modulation sidebands on the shotnoise level.* **Method 5 and 6** *also make use of the* `qshot` *detector but include some automatic scaling and conversion of the result. The small difference between the methods using the* `qshot` *detector to the first three methods comes from the fact that the applied correction of $\sqrt{2}$ in methods 1 to 3 is only approximate.*

the GEO summary pages http://www.geo600.uni-hannover.de/georeports/. From these parameters we can reconstruct approximately the optical gain of the detector for the reference time.

Now, we can simulate the optical gain with FINESSE and tune the demodulation phase such that we obtain the same transfer function. For one set of demodulation phases the input file uses the following commands:

```
fsig sig1 MCN 1 0
fsig sig2 MCE 1 180
```

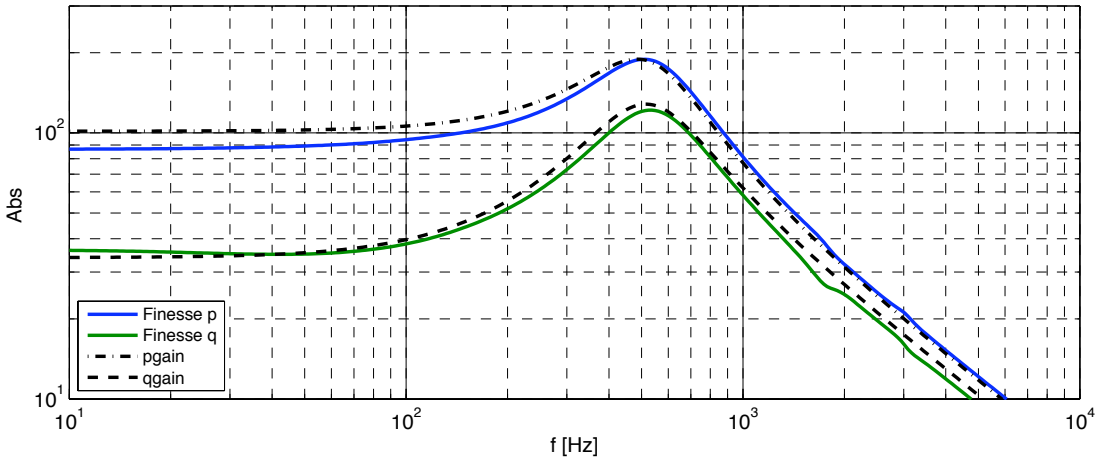


Figure B.2: *comparison of the optical gains for the P and Q channel of the dark fringe output signal. The plot shows the transfer functions reconstructed from the parameters given on the GEO summary webpage for the 03.03.2006 and the simulated transfer function for demodulation phases of 4 and 101 degrees.*

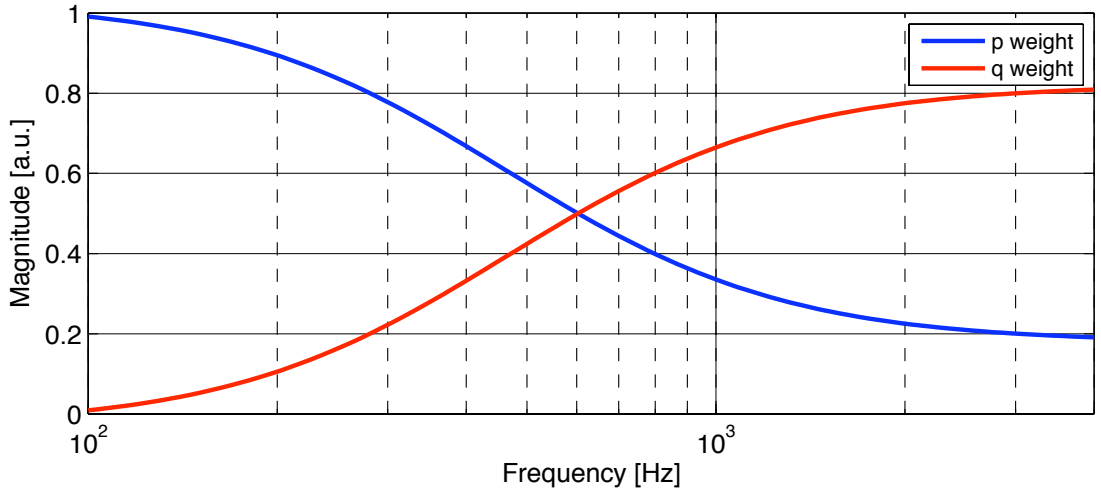


Figure B.3: *Weights for P and Q quadrature.*

```
# compute transfer function  Delta_L -> DF
pd2 pdMI1 $fMI 4 1 nMSR2
pd2 pdMI2 $fMI 101 1 nMSR2

xaxis sig1 f log 10 10k 300
put pdMI1 f2 $x1
put pdMI2 f2 $x1
```

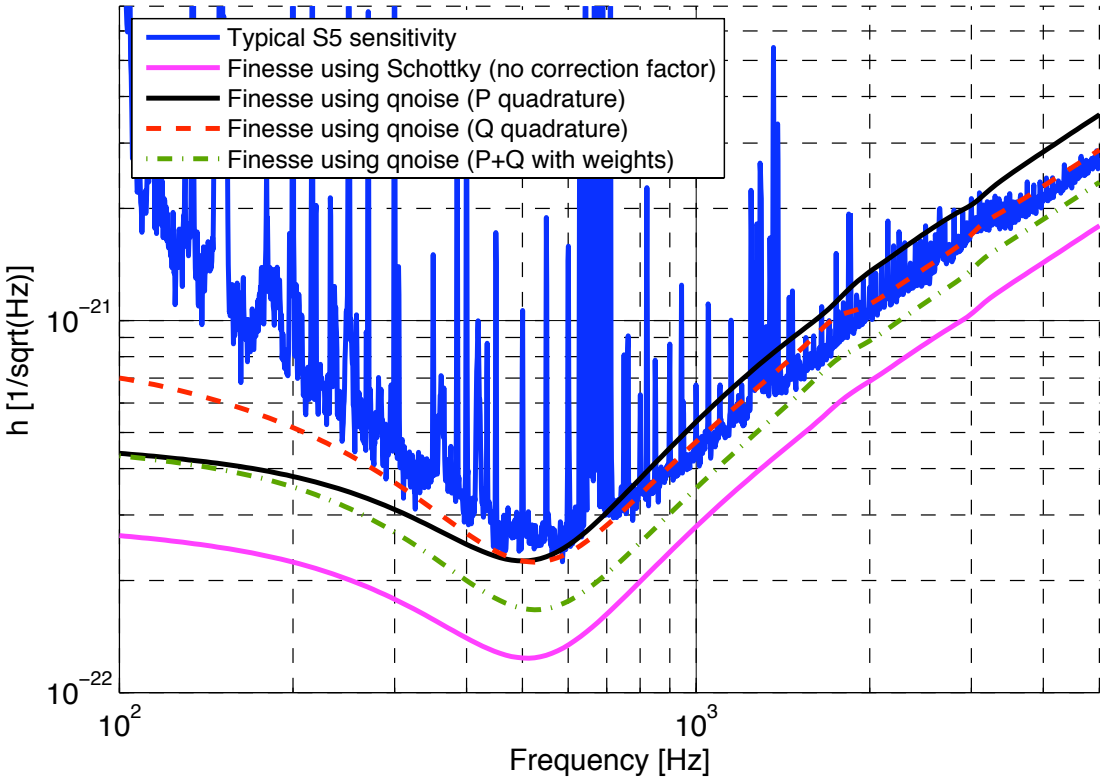


Figure B.4: *Typical S5 sensitivity compared to the* FINESSE *computation of the shot-noise-limited sensitivity.*

Figure B.2 shows a comparison of the transfer function reconstructed from the parameters given on the webpage with the simulated transfer function for demodulation phases of 4 and 101 degrees. The resulting comparison of the FINESSE model with the measured GEO 600 sensitivity is shown in Figure B.4.

Appendix C

Realistic thermal distortions in Advanced LIGO arm cavities

In the last few years FINESSE has undergone extensive development to include the effects of mirror surface distortions. The application of map surfaces has grown from a very basic integration routine (using a Riemann sum) to include multiple integration routines, which can be used to optimise the results for different geometric effects and drastically increase the speed of the coupling calculations (see sction 4.7.9). This involves extensive testing of the code, including internal tests of FINESSE map routines against analytical methods for calculating the effect of simple distortions (i.e. misalignment, see section 4.7.11) and comparisons of FINESSE results with other simulation tools using significantly different simulation methods. This has been my particular job during the development of FINESSE (Charlotte Bond). As part of this on-going task we appealed to other simulators in the gravitational wave community for simulation examples which they believed would be a good test of FINESSE. We were challenged by Hiro Yamamoto to simulate an Advanced LIGO arm cavity with thermally distorted mirrors and determine the loss of power during one round-trip of a light field in such a cavity. Previous attempts to simulate this using other modal methods have failed to replicate the results achieved using Fast-Fourier-Transform (FFT) propagation methods, such as those carried out by Yamamoto [Yamamoto]. The application of thermal effects in models of gravitational wave detectors will be crucial for the commissioning of advanced detectors, so it is vital that FINESSE can simulate these effects accurately.

The failure of previous modal models and the fact that the inclusion of thermal distortions requires delicate handling makes this setup the perfect test of FINESSE. This is also a good example with which illustrate the specific steps required to optimise a FINESSE simulation with mirror maps and achieve accurate results with relatively low `maxtem`. In this section we present a study of the losses incurred in the arm cavities of an Advanced LIGO interferometer when the mirrors are thermally distorted. The distortions of the mirrors in the Advanced LIGO arm cavities are expected to be relatively large, due to the high circulating powers expected in the arms. This study is a good test for the use of mirror maps in FINESSE, as these are important simulations for commissioning and require some effort to simulate the setup correctly.

C.1 Preparing mirror maps

In order to simulate the effects of thermal distortions in FINESSE the expected distortion of the mirrors are calculated and stored in the FINESSE mirror map format. In this case we consider the distortion after the mirror has achieved thermal equilibrium and calculate the distortions using the Hello-Vinet method [Hello-Vinet]. Any absorption of the laser beam in the mirrors results in a temperature gradient in the mirror substrate, causing the material to expand and deform as well as causing a change in the refractive index. This has two effects: 1) it distorts the surface of the mirror from an ideal curved surface; 2) it creates an effective lens in the substrate. In this study we consider only the effect of the surface distortion.

The absorption can occur in either the mirror substrate or the coating. For simplicity we here consider only the absorption in the coating and not the substrate. For this example we investigate a realistic absorption level of 1 ppm (part-per-million) per mirror. We consider three cases:

1. No absorption. The mirrors are represented as perfect spheres, the only realistic geometric effect being their finite apertures.
2. Unbalanced. 1 ppm absorption in the end test mass (ETM) and no absorption in the input test mass (ITM).
3. Balanced. 1 ppm absorption in each mirror.

For these examples we require two mirror surface maps, one for the ITM and one for the ETM. The distortions on each mirror will be slightly different, as although they have the same radius, thickness and material properties the beam spot sizes incident on the mirrors will be different: 5.3 cm on the ITM and 6.2 cm on the ETM. The temperature gradient and resulting thermal distortion depend on the size and shape of the incident beam. Several SimTools (see section 7.2.1) functions have been developed for the purposes of calculating the thermal distortions and lensing for just such a setup. The function `FT_mirror_map_from_thermal_distortion.m` was used to produce the maps for this investigation. This function employs the Hello-Vinet formula to calculate the resulting distortion of a mirror with given dimensions and thermal properties illuminated by a gaussian beam of a given spot size. In figure C.1 plots of the thermal distortion for the end test mass are shown, both as a cross-section of the mirror surface and as a mirror surface map. This distortion is dominated by low spatial frequencies and a substantial part can be described as a change in curvature of the mirror.

In order to successfully calculate the effects of mirror surface distortions in FINESSE the Gaussian beam parameter used to calculate the coupling coefficients must be carefully chosen. For the case of resonant cavities the most appropriate choice of beam parameter is commonly that which matches the geometry of the cavity, in this case using a beam parameter whose curvature matches the curvature of the mirrors. In order to apply this successfully any curvature of the mirrors should be applied in FINESSE using the `attr` command, rather than being contained in a mirror map. The `cav` command can then be used to set the Gaussian parameter to be mode-matched to the cavity. Although for

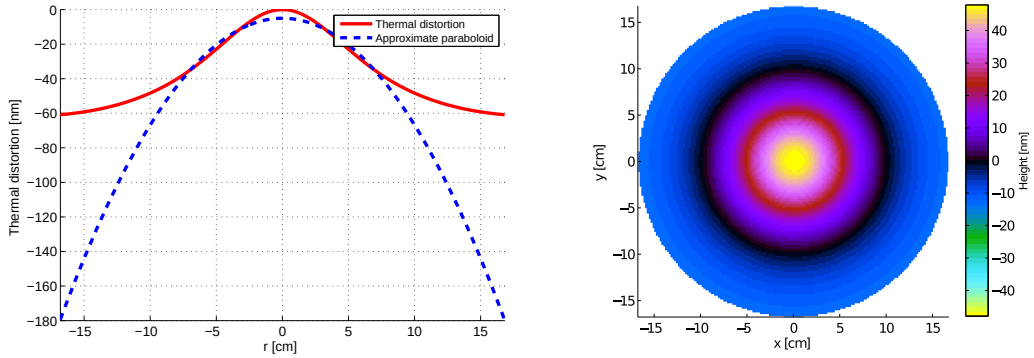


Figure C.1: *Plots showing the expected thermal distortion of the end test mass in an Advanced LIGO arm cavity with 1 ppm coating absorption. The cross section (left) shows the overall distortion and an approximate paraboloid fitted to the distortion. The equivalent mirror map is shown (right).*

small curvatures both methods should give equivalent results, a good choice of Gaussian parameter should require fewer higher order modes and a lower `maxtem`. Therefore, for this investigation the curvatures are removed from the surface maps before they are applied to the mirrors. Again, several SimTools functions have been developed for just such a task. Tools using Zernike polynomials have been produced for the analysis and preparation of maps, analysing the entire surface over the defined mirror disc. However, this particular case involves fitting a curved surface to the map and then removing it (`FT_remove_curvature_from_mirror_map.m`). As the mirrors are not uniformly illuminated the curvature fitting process should be most accurate where the beam is at its most intense, i.e. the fitting process should be weighted by the appropriate gaussian beam. This is achieved by minimising:

$$x = \int_0^a \int_0^{2\pi} W(r)\,[S_{\mathrm{map}} - S_{\mathrm{C}}]^2\, r\, \mathrm{d}\phi\, \mathrm{d}r$$

where S_{map} is the surface discribed by the map, S_C is the fitted curved surface (described by a radius of curvature, R_{c}) and $W(r)$ is the weighting function. In this case $W(r)$ is the gaussian beam intensity function, with a given spot size. For the case of mirror thermal distortions the weighted curvature (so-called *approximate paraboloid*) can be calculated analytically [VPB] using the SimTools function `FT_approximate_paraboloid_for_thermal_distortion.m`. The cross-section of the approximate paraboloid found using this function is plotted in figure C.1 for the end test mass, a curved surface with $R_{\mathrm{c}} = -80\,\mathrm{km}$. The curved surface is removed from the map, the residual distortion of the ETM is shown in figure C.2. The distortion of the ITM was similarly calculated and a weighted curvature of $R_{\mathrm{c}} = -60\,\mathrm{km}$ was removed.

In most cases this removed curvatures would then be included in the radius of curvature defined in the FINESSE script (using the `attr` command). However, for this investigation we assume that the Advanced LIGO thermal compensation system is working as expected,

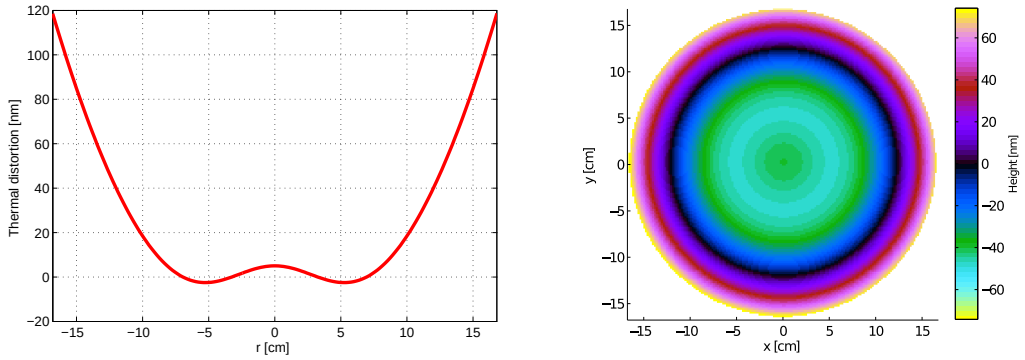


Figure C.2: *Plots showing the cross section (left) and mirror map (right) of the expected thermal distortion of the end test mass in an Advanced LIGO arm cavity, where a Gaussian weighted curvature has been fitted and removed from the surface.*

correcting for the curvature change caused by the thermal deformation. Therefore, we assume the curvatures are corrected back to their nominal values, 2245 m (ETM) and 1934 m (ITM). Generally, when using mirror maps, we would also fit and remove any tilt of the surface, effectively aligning the mirror in the simulation setup. However, in this case the thermal distortions contain no tilt term, so we can omit this step in the map preparation. We do, however, remove any offset from the maps using the SimTools function `FT_remove_offset_from_mirror_map.m`. Any overall offset of the mirror should be set in the FINESSE file by tuning the mirror position, not hidden in the mirror map.

C.2 Simulation setup

The simulation setup is based on the simple Fabry-Perot cavity. The mirror parameters and cavity length are the Advanced LIGO design parameters:

```
m mITM 0.985965 0.014 0 nITM2 nITM1
s sC 3994.5 nITM2 nETM1
m mETM 0.99996 5u 0 nETM1 dump
```

The curvature of the mirrors are defined using the `attr` command and the beam injected into the cavity is mode matched to the cavity using the `cav` command:

```
attr mITM Rc 1934
attr mETM Rc 2245
cav armcav mITM nITM2 mETM nETM1
```

Both mirrors are seen as concave from inside the cavity. In FINESSE the sign of the radius of curvature is related to the order of the nodes in the mirrors definition. As both mirrors are defined with the node inside the cavity first (`nITM2` and `nETM1`) the curvatures are given as positive. For all 3 cases the finite aperture (16.8 cm) of the mirrors must be

specified using the `attr` command:

```
attr mITM r_ap 0.168
attr mETM r_ap 0.168
```

Finally the mirror maps must be applied to the ITM (case 3) and to the ETM (case 2 and 3). For example, to include the map describing the thermal distortion of the ETM, the following commands are required:

```
# ETM map commands
map  mETM etm_thermal_res_map.txt
knm  mETM etm_map_coupling
conf mETM save_knm_binary 1
conf mETM interpolation_method 2
conf mETM integration_method 3
```

The `map` command specifies the file containing the mirror map stored in the standard FINESSE format. A file, "*etm_map_coupling*" is specified in which to save the coupling coefficients, in binary form for speed of access (`save_knm_binary`). We also specify the linear interpolation of the mirror surface and the cuba parallel integration method for the calculation of the coupling coefficients (see section 4.7.9). The ITM map is applied with equivalent commands. However, care should be taken in the case of the ITM. In FINESSE the order of the nodes specifies which way the map is applied to the mirror. The z-axis of the map surface will point towards the first specified node, away from the second node. Therefore, to orientate the ITM map with the surface facing the inside of the cavity the order of the nodes should be specified with the cavity node first.

C.3 Results

For this investigation the figure of merit we have chosen is the round-trip loss incurred for the 3 different cases. This is a useful single number for comparison between different simulation methods. Previous attempts with other modal based methods have failed to agree with other methods in similar investigations of the losses in thermally distorted cavities. This makes this investigation a good test of FINESSE as a robust, accurate tool for calculating the effects of realistic optics.

In FINESSE a single round-trip of a cavity is not simply simulated. The round-trip loss is therefore calculated using the power circulating in the cavity:

$$L_{\mathrm{arm}} = T_{\mathrm{ITM}} \left(\sqrt{\frac{P_{\mathrm{FP}}(0)}{P_{\mathrm{FP}}(L_{\mathrm{arm}})}} - 1 \right) \tag{C.1}$$

where T_{ITM} is the transmission coefficient of the input mirror, $P_{\mathrm{FP}}(0)$ is the circulating power in the equivalent, lossless cavity and $P_{\mathrm{FP}}(L_{\mathrm{arm}})$ is the power circulating in the lossy cavity.

In FINESSE the setups for case 1, 2 and 3 were simulated for a range of maxtem, from 0 to 20. Due to the relatively large nature of the thermal distortions it is expected that we will require a relatively high `maxtem`. For comparison the same simulations were also carried out using a Fast-Fourier Transform (FFT) method, based on OSCAR [OSCAR]. The circulating power is detected when the cavity is tuned to the point where the power is at a maximum. Locking sequences and more compex operating points are omitted in this example as these require delicate handling in the FFT code. From the circulating power the round-trip loss is calculated for each case. Figure C.3 shows plots of the round-trip losses as the `maxtem` is increased for cases 2 and 3.

These results, for the cases where thermal distortions are applied to the mirrors, demonstrates the importance of including the right number of higher-order modes in FINESSE simulations. A lot of coupling into higher order modes occurs at lower orders, but the mid orders (6-10) still contribute significantly to the overall result. Choosing too small a `maxtem` can lead to wildly inaccurate results. However, by including enough higher-order modes FINESSE can successfully re-create results achieved using other methods, with the added advantage that once the coupling coefficients are calculated the effects can be incorporated into the simulations without the need for re-calculation.

The results for specific values of `maxtem` for the 3 cases are summarised in table C.1. For case 1 the round-trip loss is incurred due to the finite size of the mirrors and the power clipped by these apertures. The aperture is large, compared to the size of the beam, and therefore the coupling into higher order modes is small. Therefore, the round-trip loss stabilises with low `maxtem`, agreeing well with the result of the FFT propagation method. For case 2 the distortion of the end mirror causes coupling into higher order modes, specifically modes of an even order as the distortions are symmetric, spherical aberrations. Although the most significant coupling happens at low orders there is still

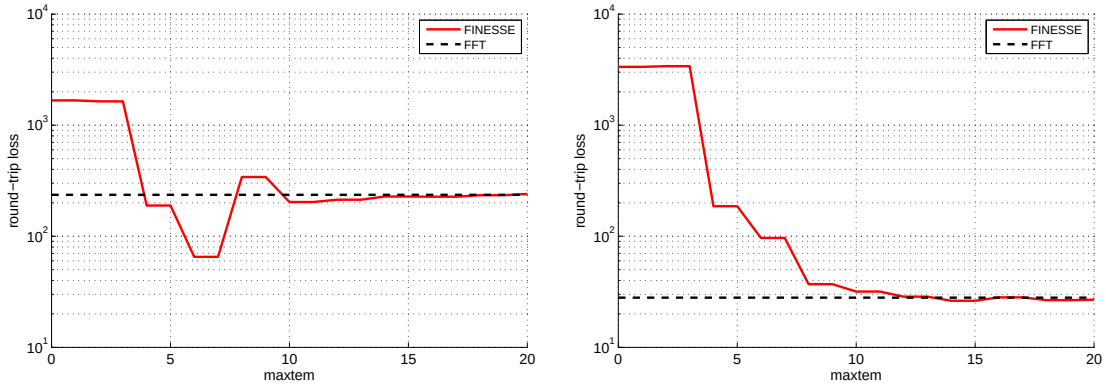


Figure C.3: *Plots showing the round-trip loss for different values of maxtem in Advanced LIGO cavities with thermal mirror distortions simulated in* FINESSE *and using an FFT propagation method. The effect of thermal distortions are added to the end mirror (left) and both mirrors (right).*

a relatively large contribution from the higher orders. At a `maxtem` of around 8-10 the round-trip loss reaches the right magnitude and by `maxtem` 15 the FINESSE result agrees very well with the FFT result. Another interesting point regarding case 2 is that the addition of higher order modes does not just account for the power previously disregarded in higher-order modes. For example if we were to simulate such a setup using only `maxtem` 6 the round-trip loss would be significantly underestimated. This is because this example deals with a cavity, where any coupling during one mirror incidence results in modes which can couple into higher order modes.

Finally, in case 3 the round-trip loss is actually greatly reduced from the case where we consider just one distorted mirror. As the distortions on the two mirrors are identical, as scaled by the incident beam size, the distortion of the wavefront by the end mirror will match the distorted input mirror. This match of the distorted mirror and wavefront causes much less subsequent distortion of the beam, less coupling into higher-order modes and hence less power clipped by the finite size of the mirrors. This result is recreated in the FINESSE simulation, an effect which previous modal models were unable to achieve, using `maxtem` 10-15.

Case	FINESSE (maxtem 10)	FINESSE (maxtem 15)	FFT
1	0.8	0.9	0.9
2	203	228	234
3	32	26	27

Table C.1: *Round-trip losses in simulated Advanced LIGO cavities for three different cases: 1) no thermal distortions; 2) a thermally distorted end mirror; 3) thermal distortions of the input and end mirrors. The cavities are simulated using an FFT propagation method and* FINESSE.

For either case we have demonstrated that we can achieve similar results using FINESSE with a finite number of modes. In particular, `maxtem` 10 is used to achieve the correct order of magnitude, which should be good enough in this case as simulation and experiment will not match down to exact ppm losses. However, a more accurate result can be achieved using a `maxtem` of around 15.

One final consideration is the accuracy we are using in the calculation of the coupling coefficients. This is defined in the "*kat.ini*" file by the variable `abserr`, the absolute error of the coupling coefficients. The results summarised above were achieved using an `abserr` of 1×10^{-6}. The default value set in the "*kat.ini*" file is 1×10^{-3}. The results achieved using this default value are summarised in table C.2. The do not differ by a large degree to those using 10^{-6}

Case	FINESSE (maxtem 10)	FINESSE (maxtem 15)
1	0.7	0.8
2	202	225
3	33	27

Table C.2: *Round-trip losses in Advanced LIGO cavities for three cases: 1) no thermal distortions; 2) a thermally distorted ETM; 3) thermal distortions of the ITM and ETM. The cavities are simulated using* FINESSE *and the default absolute error of* 1×10^{-3}.

Appendix D

Further reading: Finesse in practice

The interested reader is directed to the following documents, which give extensive details on testing different Finesse features and specific commissioning tasks for Advanced LIGO[1].

Testing:

- [Bond13] Interferometer responses to gravitational waves are derived, from simple spaces to Michelson and Sagnac interferometers, and compared with the responses produced using Finesse.
- [Clarke] The improved modelling of *sidebands of sidebands* in Finesse is described.
- [Ballmer] Comparisons of alignment signals calculated for Fabry-Perot cavities using three methods: Finesse, an analytic calculation and the FFT propagation simulation OSCAR [OSCAR].
- [Bond13b] Comparisons of the control signals and sideband build up in Advanced LIGO, as modelled in Finesse and Optickle.

Commissioning:

- [Dooley] Report on various commissioning tasks and particular simulation issues, as covered in the Advanced LIGO commissioning meeting, January 2013.
- [Bond13c] A dedicated commissioning investigation into power loss at the central beam-splitter in Advanced LIGO using Finesse.
- [Kokeyama] Finesse simulations of the alignment control signal of the Advanced LIGO input mode cleaner.
- [Bond14] Simulations of the effects of mode mismatches on longitudinal control signals at the Livingston Advanced LIGO interferometer.

Finesse files:

- [L1kat] Finesse files for the Advanced LIGO interferometer at Livingston.
- [H1kat] Finesse files for the Advanced LIGO interferometer at Hanford.

[1] I believe that similar documents exist also in other projects, this list is simply a collection of documents related to our own modelling work.

Appendix E

Maps and Coupling Coefficients

The scenario often arises where we want to apply misalignments, mode mismatches, a surface distortion and an aperture to simulate a realistic mirror. The coupling coefficient is more commonly refered to as an inner product or projection of a beam from one 'basis set' into another. The basis sets we are considering here are the orthogonal Hermite-Gaussian (HG) functions. A beam at any point can be decomposed into a HG basis set and is described by a vector $a_{in,N}, N \in \{0, 1, 2, \ldots, N_{max}\}$, where N_{max} is the highest order of HG polynomial we consider. It is also assumed there exists some linear mapping between the HG mode number in the sagittal and transverse directions $nm \rightarrow N$ and $n'm' \rightarrow M$, to save writing nm and $n'm'$ all the time.

The outgoing beam of mode M has the beam parameter q_2, the incoming mode N is q_1 – these beam parameters are known values and are set by the user (and FINESSE's beam tracing routine). However the incoming beam must first interact with the mirror's or beamsplitter's known properties, i.e. its radius of curvature represented by the application of the ABCD matrix which transforms the incoming beam as $q_1 \rightarrow q_1'$. The projection between q_1' to the basis q_2 is then required. This is given by the coupling coefficient integral (also known as an overlap integral or complex function space inner product) between the incoming and outgoing beam basis. The coupling between mode N and M is computed as

$$k_{NM} = \langle U_N, U_M \rangle = \iint_{-\infty}^{\infty} U_N(x, y, q_1') U_N^*(x, y, q_2) dx dy \tag{E.1}$$

This so far is just computing the mode-mismatch between q_1'. The values of k_{NM} can be arranged in the form of a matrix

$$K = \begin{bmatrix} \langle U_0, U_0 \rangle & \langle U_1, U_0 \rangle & \langle U_2, U_0 \rangle & \ldots & \langle U_{N_{max}}, U_0 \rangle \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \langle U_0, U_{N_{max}} \rangle & \langle U_1, U_{N_{max}} \rangle & \langle U_2, U_{N_{max}} \rangle & \ldots & \langle U_{N_{max}}, U_{N_{max}} \rangle \end{bmatrix} \tag{E.2}$$

Which can then be applied to the mode content vector of the incoming beam to get the outgoing beam mode content in the beam parameter q_2

$$a_{out} = K a_{in} \tag{E.3}$$

The coupling coeffcient equation can also tell us if we further distort the incoming beam, for example via a mirror surface map, how this will project into the outgoing beam. Here,

for example, A and B are two separate distortions, such as a tilt of the optic surface and some astigmatic surface curvature. Then the overlap integral becomes

$$k_{NM} = \iint_{-\infty}^{\infty} U_N(x, y, q_1')A(x, y)B(x, y)U_N^*(x, y, q_2)dxdy \tag{E.4}$$

This can be represented in multiple ways using the inner product notation

$$\begin{aligned} \iint_{-\infty}^{\infty} U_N(x, y, q_1')A(x, y)B(x, y)U_N^*(x, y, q_2)dxdy &= \langle U_N AB, U_M \rangle & \text{(E.5)} \\ &= \langle U_N A, B^* U_M \rangle & \text{(E.6)} \\ &= \langle U_N, A^* B^* U_M \rangle & \text{(E.7)} \end{aligned}$$

Assuming that the effect of B is something simple and (on its own) could be implemented using a analytic solution and A is complicated and requires a numerical integration that uses much computational power. This arises, for example, when we have some real measurements of an optics surface which is represented as a surface map, and at the same time we also may want to tilt the optic by a range of different angles to generate alignment signa. The latter requires many recomputations of K, but we posses analytic solutions in the form of the Bayer-Helms routines. It the current form the entire matrix K would need to be recomputed even though the mirror surface map remains unchanged.

To circumnavigate this problem we can try to separate the effects into two sets of coupling coefficients. Mathematically this is done by "inserting unity" – for lack of a better term – into the coupling coefficient equation

$$k_{NM} = \sum_{L}^{\infty} \langle U_N A, U_L \rangle \langle U_L, B^* U_M \rangle . \tag{E.8}$$

This can be understood better perhaps in terms of dot products of vectors, which is a more specialised inner product. Take the vectors a and b of equal length which are then projected onto the infinite basis vector set c

$$\begin{aligned} a &= \sum_n A_n c_n & \text{(E.9)} \\ b &= \sum_n B_n c_n & \text{(E.10)} \\ \langle a, b \rangle &= \sum_n a_n \cdot b_n & \text{(E.11)} \\ &= \sum_l^{\infty} (a \cdot c_l)(c_l \cdot b) & \text{(E.12)} \end{aligned}$$

Using this trick for the HG coupling coefficients we create 2 inner products: one that can be solved analytically and another which requires numerical integration; we essentially

need to compute 2 separate K matrices.

$$K_{NM} = \sum_{L}^{\infty} \langle U_N A, U_L \rangle \langle U_L, B^* U_M \rangle = [K_A K_B]_{NM} \tag{E.13}$$

$$= \sum_{L}^{\infty} \iint_{-\infty}^{\infty} U_N(x, y, q_1') A(x, y) U_L^*(x, y, q_L) dx dy \tag{E.14}$$

$$\ldots \iint_{-\infty}^{\infty} U_L(x, y, q_L) B(x, y) U_M^*(x, y, q_2) dx dy \tag{E.15}$$

Of course our choosing the beam parameter that U_L represents is arbritary here. However, the choice U_L will have a strong influence on the numerical error in practise because only a limited set of modes is used in the simulation whereas the equation E.8 is only correct if the complete set is used. The two choices that should provide best performance in most cases is to use either q_1' or q_2. It is also worth nothing that the positioning of A and B in the inner products is also completely arbritary, in the sense we could also compute

$$K_{NM} = \sum_{L}^{\infty} \langle U_N AB, U_L \rangle \langle U_L, U_M \rangle \tag{E.16}$$

however the second inner product here is just δ_{LM} (if $q_L = q_2$), which offers no computational improvement over what we had originally. If $q_L = q_1'$ values differ we are essentially moving the mode-mismatch computation into a separate matrix and computing the distortion coupling back into the same beam basis.

Our outgoing beam is now computed as

$$a_{out} = K a_{in} = (K_A K_B) a_{in} \tag{E.17}$$

Commutations should not exist as long as enough modes are used in the projection to U_L as the ordering of A and B is arbitrary. An issue may arise if the solver chosen for K_A or K_B is an analytic solution will be much more accurate $\approx 10^{-15}$ than a numerical solver which are typically 10^{-6}, thus commutation errors will arise from this due to these errors.

E.1 Correct implementation

From the previous analysis we can now state the optimal routine for separating the true coupling coefficient matrix K. For distortions A and $B$the coupling from mode N to mode M whose beam parameters are respectively q_1' and q_2 is

$$K_{NM} = [K_A K_B]_{NM} \tag{E.18}$$

$$= \sum_{L}^{\infty} \underbrace{\langle U_N(q_2) A(x, y), U_L(q_L) \rangle}_{\text{A Solver}} \underbrace{\langle U_L(q_L), B^*(x, y) U_M(q_2) \rangle}_{\text{B Solver}}. \tag{E.19}$$

We further defined the solver as follows: if a distortion (A or B) is a surface map, the respective matrix is to be computed by numerical integration. In all other cases the matrix is computed using the Bayer-Helms equations.

We provide to user command to influence the ordering and the selction of q_L. The order of the matrix calculation is defined by

```
conf [component] knm_order AB [A,B = 1(Map) or 2(Bayer-Helms)]
```

where the argument states which solver and distortion is applied to which coupling coefficient matrix K_A and K_B. The value of q_L also needs to be decided, here we use the command

```
conf [component] knm_change_q [1 or 2]
```

where if the argument is `1` then $q_L = q_1'$ else if `2` then $q_L = q_2$. The command `knm_apply_ABCD` is no longer needed.

E.2 Separating more distortions

Although not necessary now, in the future the need to compute more than two distortions efficiently might be needed. The method for adding more is simply a repetition of the previous steps splitting the initial inner product into two. Inner product A or B must be chosen to be split, I will choose B here arbitarily to add a distortion C

$$
\begin{aligned}
K_{NM} &= \langle U_N(q_2) A(x,y) B(x,y) C(x,y), U_M(q_2)\rangle && \text{(E.20)}\\
&= \sum_L^{\infty} \underbrace{\langle U_N(q_2) A(x,y), U_L(q_L)\rangle}_{\text{A Solver}} \underbrace{\langle U_L(q_L) B(x,y) C(x,y), U_M(q_2)\rangle}_{\text{B Solver}} && \text{(E.21)}\\
&= \sum_L^{\infty} \underbrace{\langle U_N(q_2) A(x,y), U_L(q_L)\rangle}_{\text{A Solver}} && \text{(E.22)}\\
&\quad \ldots \sum_J^{\infty} \underbrace{\langle U_L(q_L) B(x,y), U_J(q_J)\rangle}_{\text{B Solver}} \underbrace{\langle U_J(q_J), C^*(x,y) U_M(q_2)\rangle}_{\text{C Solver}} && \text{(E.23)}\\
&= [K_A (K_B K_C)]_{NM} && \text{(E.24)}
\end{aligned}
$$

Here we now have another matrix K_C which we need to compute, this requires another projection onto the basis U_J whose beam parameter is q_J.

E.3 Coupling coefficient integration performance improvements

Calculating the coupling coefficients by numerical integration is a computationally expensive process and can last from minutes to days depending on the map and incoming/outgoing beam parameters. As seen previously the coupling coefficient matrix can be

split up into 2 separate ones for both the numerical integration result and Bayer-Helms. We also had to make a choice for the value of q_L, if this is chosen so that the numerical integral matrix has the same incoming and outgoing beam parameter an $\approx \times 2$ speedup can be achieved. Taking the coupling coefficient integral we get

$$\begin{aligned} U_{nm} &= (2^{n+m-1}n!m!\pi)^{-1/2}\frac{1}{w(z)}e^{i\psi(z)(n+m+1)}... \\ &\quad ...H_n\left(\frac{\sqrt{2}x}{w_{o,x}}\right)H_m\left(\frac{\sqrt{2}y}{w_{o,y}}\right)e^{-i\frac{kr^2}{2q(z)}} \qquad \text{(E.25)} \\ &= \frac{A_{nm}}{w(z)}e^{i\psi(z)(n+m+1)}H_nH_me^{-i\frac{kr^2}{2q(z)}} \qquad \text{(E.26)} \\ k_{nmn'm'} &= \int U_{n'm'}F(x,y)U^*_{nm}dA \qquad \text{(E.27)} \\ k_{nmn'm'} &= \frac{A_{nm}A_{n'm'}}{w_{out}(z)w_{in}(z)}e^{i\psi_{out}(n'+m'+1)}e^{-i\psi_{in}(n+m+1)}... \\ &\quad ...\int F(x,y)H_nH_mH_{n'}H_{m'}e^{-i\frac{kr^2}{2}\left(\frac{1}{q_{out}(z)}-\frac{1}{q^*_{in}(z)}\right)}dA \qquad \text{(E.28)} \end{aligned}$$

The interesting case for us here is when $q_{in} = q_{out}$, if this is true then the above simplifies,

$$\begin{aligned} k_{nmn'm'} &= \frac{A_{nm}A_{n'm'}}{w^2(z)}e^{i\psi(\Delta n+\Delta m)}\int F(x,y)H_nH_mH_{n'}H_{m'}e^{-\frac{x^2+y^2}{w^2(z)}}dA, \qquad \text{(E.29)} \\ \Delta n &= n'-n, \qquad \text{(E.30)} \\ \Delta m &= m'-m. \qquad \text{(E.31)} \\ & \qquad \text{(E.32)} \end{aligned}$$

We then find that the transpose elements are nearly identical, i.e. $00 \to 10$ and $10 \to 00$, expect for the an opposite sign in the Gouy phase

$$k_{n'm'nm} = \frac{A_{nm}A_{n'm'}}{w^2(z)}e^{-i\psi(\Delta n+\Delta m)}\int F(x,y)H_nH_mH_{n'}H_{m'}e^{-\frac{x^2+y^2}{w^2(z)}}dA. \qquad \text{(E.33)}$$

Dividing one by the other leaves us with the final relationship between the elements in the matrix

$$\begin{aligned} k_{n'm'nm} &= k_{nmn'm'}e^{-i2\psi(\Delta n+\Delta m)}, \qquad \text{(E.34)} \\ |k_{n'm'nm}| &= |k_{nmn'm'}|. \qquad \text{(E.35)} \end{aligned}$$

So we see that the matrix is symmetric if Gouy phase is 0, as it is at the beam waist. This is a useful relationship especially for calculating the coupling matrices as now we need only calculate one half of the matrix. The transpose elements can easily be found by just multiplying by the phase factor. The potential computational reduction increases when using more modes if mode matched by a factor $(N-1)/2N$, where N is the number of modes used.

This performance increase is on by default and switches off in the code if not mode-matched. It can be manually controlled from the `kat.ini` file with the option

`calc_knm_transpose [0 or 1].`

Appendix F

Some mathematics

This appendix gives some details about some of the formulae and algorithms used in FINESSE.

F.1 Hermite polynomials

The first few Hermite polynomials in their unnormalized form can be given as:

$$\begin{aligned} H_0(x) &= 1, & H_1(x) &= 2x, \\ H_2(x) &= 4x^2 - 2, & H_3(x) &= 8x^3 - 12x. \end{aligned} \tag{F.1}$$

Further polynomial orders can be computed recursively using the following relation:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x). \tag{F.2}$$

In FINESSE the functions up to H_{10} are hard-coded and for higher orders the recursion relation is used.

F.2 The paraxial wave equation

An electromagnetic field (at one point in time, in one polarisation, and in free space) can in general be described by the following scalar wave equation [Siegman]:

$$\left[\nabla^2 + k^2\right] E(x, y, z) = 0. \tag{F.3}$$

Two well-known exact solutions for this equation are the plane wave:

$$E(x, y, z) = E_0 \ \exp\left(-\mathrm{i}\, kz\right), \tag{F.4}$$

and the spherical wave:

$$E(x, y, z) = E_0 \ \frac{\exp\left(-\mathrm{i}\, kr\right)}{r} \qquad \text{with} \qquad r = \sqrt{x^2 + y^2 + z^2}. \tag{F.5}$$

Both solutions yield the same phase dependence along an axis (here, for example, the z-axis) of $\exp(-\mathrm{i}\, kz)$. This leads to the idea that a solution for a beam along the z-axis can

be found in which the phase factor is the same while the spatial distribution is described by a function $u(x, y, z)$ which is slowly varying with z:

$$E(x, y, z) = u(x, y, z) \ \exp(-\mathrm{i}\, kz). \tag{F.6}$$

Substituting this into Equation F.3 yields:

$$\left(\delta_x^2 + \delta_y^2 + \delta_z^2\right) u(x, y, z) - 2\mathrm{i}\, k\delta_z u(x, y, z) = 0. \tag{F.7}$$

Now we put the fact that $u(x, y, z)$ should be slowly varying with z in mathematical terms. The variation of $u(x, y, z)$ with z should be small compared to its variation with x or y. Also the second partial derivative in z should be small. This can be expressed as:

$$\left|\delta_z^2 u(x, y, z)\right| \ll \left|2k\delta_z u(x, y, z)\right|, \left|\delta_x^2 u(x, y, z)\right|, \left|\delta_y^2 u(x, y, z)\right|. \tag{F.8}$$

With this approximation, Equation F.7 can be simplified to the *paraxial wave equation*:

$$\left(\delta_x^2 + \delta_y^2\right) u(x, y, z) - 2\mathrm{i}\, k\delta_z u(x, y, z) = 0. \tag{F.9}$$

Appendix G

Syntax reference

In order to use the program you have to know and understand the commands for the input files. The following paragraphs give a full explanation of the syntax. The help screens (use 'kat -h' or 'kat -hh') give a short syntax reference. See other our online syntax reference at http://www.gwoptics.org/finesse/reference/.

G.1 Comments

Two different methods are available for adding comments to the FINESSE input files. First, each line can be 'commented out' by putting a single comment sign at the start of the line. The comment signs are `#`, `"` and %.

Any of these signs can also be used to add a comment at the end of a line, for example:
`xaxis mirror phi lin -20 20 100 # tune mirror position`

The second commenting method is the use of C-style block comments with `/* - - - */`. This is very useful for including several different sets of commands in one input file. In the following example some temporarily unused photodiodes have been commented out:

```
/*
pd pd1 n23
pd pd2 n24
pd pd3 n25
pd pd4 n26
*/

pd1 pd1p 1M 0 n23
```

Please note that these comment strings must be used in empty lines, otherwise the (very simple) parser cannot handle them correctly.

G.2 Components

Parameters in square brackets [ ] are optional.

- **m** : mirror
  usage : `m name R T phi node1 node2`
  R = power reflectivity (0<R<1)
  T = power transmittance (0<T<1)
  phi = tuning in degrees
  A positive tuning moves the mirror from `node2` towards `node1`.

- **m1** : mirror
  usage : `m1 name T Loss phi node1 node2`
  T = power transmittance (0<T<1)
  Loss = power loss (0<Loss<1)
  phi = tuning in degrees
  Note: the values are not stored as T and L but as R and T with 0<R, T<1. Thus, only R and T can be tuned (e.g. with `xaxis`).

- **m2** : mirror
  usage : `m2 name R Loss phi node1 node2`
  R = power reflectivity (0<R<1)
  Loss = power loss (0<Loss<1)
  phi = tuning in degrees
  Note: the values are not stored as T and L but as R and T with 0<R, T<1. Thus, only R and T can be tuned (e.g. with `xaxis`).

- **s** : space
  usage : `s name L [n] node1 node2`
  L = length in metres
  n = index of refraction
  (default is 1.0 or specified with n_0 in 'kat.ini')

- **bs** : beam splitter
  usage : `bs name R T phi alpha node1 node2 node3 node4`
  R = power reflectivity (0<R<1)
  T = power transmittance (0<T<1)
  phi = tuning in degrees
  alpha = angle of incidence in degrees
  A positive tuning moves the beam splitter along from `node3`/`node4` towards `node1`/`node2` along a vector perpendicular to the beam splitter surface (i.e. the direction depends upon 'alpha').

- **bs1** : beam splitter
  usage : `bs1 name T Loss phi alpha node1 node2 node3 node4`
  T = power transmittance (0<T<1)
  Loss = power loss (0<Loss<1)
  phi = tuning in degrees
  alpha = angle of incidence in degrees
  Note: the values are not stored as T and L but as R and T with 0<R, T<1. Thus only R and T can be tuned (e.g. with `xaxis`).

- **bs2** : beam splitter
  usage : `bs2 name R Loss phi alpha node1 node2 node3 node4`
  R = power reflectivity (0<R<1)
  Loss = power loss (0<Loss<1)
  phi = tuning in degrees
  alpha = angle of incidence in degrees
  Note: the values are not stored as T and L but as R and T with 0<R, T<1. Thus only R and T can be tuned (e.g. with `xaxis`).

- **gr** : grating
  usage : `gr[n] name d node1 node2 [node3 [node4]]`
  d = grating period in [nm]
  Other parameters of the grating (some **must** be set) can be set via the `attr` command; these are:
  - power coupling efficiencies: eta_0, eta_1, eta_2, eta_3, rho_0
  - angle of incidence: alpha
  - radius of curvature: Rcx, Rcy, Rc (not yet implemented)

  Grating types are defined via their number of ports:
  2 1st order Littrow (eta_0, eta_1)
  3 2nd order Littrow (eta_0, eta_1, eta_2, rho_0)
  4 not Littrow (eta_0, eta_1, eta_2, eta_3)

- **isol** : isolator
  usage : `isol name S node1 node2 [node3]`
  S = power suppression in dB
  The light passes the isolator unchanged from `node1` to `node2` but the power of the light going from `node2` to `node1` will be suppressed:

$$a_{\text{out}} = 10^{-S/20}\, a_{\text{in}},$$

  with a as the field amplitude.
  When using the optional third node all the suppressed power going from node 2 to 1 is output into node3.

- **l** : laser (input light)
  usage : `l name P f [phase] node`
  P = light power in Watts
  f = frequency offset to the default frequency f_0
  (default frequency determined from 'lambda' in 'kat.ini')
  phase = phase
  Standard laser input.

- **sq** : squeezed vacuum input
  usage : `sq name f r angle node`
  f = frequency offset to the default frequency to squeeze f_0
  r = squeezing in decibels, r > 0
  angle = squeezing angle in degrees
  A squeezed vacuum input source that has variable squeezing in dB and squeezing angle. A squeezing of 20 dB should reduce the noise by a factor of 10.

- **pd** : photodiode (plus one or more mixers)
  usage : `pd[n] name [f1 [phase1 [f2 [phase2 [...] ] ] ] ] node[*]`
  n = number of demodulation frequencies ($0 \leq n \leq 5$)
  f1 = demodulation frequency of the first mixer in Hz
  phase1 = demodulation phase of the first mixer in degrees
  f2 = demodulation frequency of the second mixer in Hz
  phase2 = demodulation phase of the second mixer in degrees
  ...
  The photodetector generally computes the laser power in an interferometer output. With the command `scale ampere` the value can be scaled to photocurrent.
  Note: the number of frequencies (n) **must** be given correctly. The square brackets may be misleading here, since the parameter is not optional but can be omitted only if the number of frequencies is zero. Some likely examples are:
  `pd detector1 nout1` (or `pd0 detector1 nout1`) : DC detector
  `pd1 detector2 10M 0 nout2` : one demodulation
  `pd2 detector3 10M 0 100 0 nout2` : two demodulations
  All frequencies are with respect to the zero frequency f_0 set by 'lambda' in the init file 'kat.ini' (see Section 2.2).
  The phases are the *demodulation phases* and describe the phase of the local oscillator at the mixer. If the last phase is omitted the output resembles a network analyser instead of a mixer. This differs from a mixer because the resulting signal does contain phase information. A mixer with a fixed demodulation phase is usually used for calculating error signals whereas one often wants to know the phase of the signal for frequency sweeps, i.e. for calculating transfer functions.
  The keyword '`max`' can be used in place of the fixed demodulation phase, e.g.:
  `pd2 detector1 10M max 200 max nout1`
  This does use the optimum demodulation phase **for each data point indepen-**

dently. This can be useful when the 'best' demodulation phase is not yet known but in some circumstances it will give 'strange' output graphs. **Note that this kind of calculation does not represent any meaningful way of handling real output signals. It was merely added for convenience.**
Again, the optional asterisk behind the node name changes from the default beam to the second beam present at this node (see Section 3.2.1 for the definition of the default beam).
The positive and negative signal frequency variables, fs and mfs, can also be used in the command instead of specifying an exact frequency.

- **pdS** : shot noise limited sensitivity
  Usage is the same as for `pd`. It calculates the shot noise in the output using the DC photocurrent (equivalent to the `shot` command output) and divides it by a signal. For example,
  `pdS2 name 10M 90 100 0 n2`
  would be :
  shot noise(pd n2) / (pd2 name 10M 90 100 0 n2)
  i.e. the shot noise at node n2 divided by the signal at 100 Hz (phase= 0°) in the photocurrent at n2 demodulated at 10 MHz (phase= 90°).
  Note: This detector relies on a simple approximation for the shotnoise and only gives the correct result when no modulation sidebands are present, see Section 6.2.
  For this sensitivity output, all demodulation phases have to be set.
  If the output is a transfer function and `fsig` was used to add signals to mirrors or beam splitters it can be further normalised to m/$\sqrt{\text{Hz}}$ by `scale meter` (see below).

- **pdN** : photocurrent normalised by shot noise
  Usage is the same as for `pd` or `pdS`. It calculates the inverse of `pdS` which gives the signal to shot noise ratio.
  Note: This detector relies on a simple approximation for the shotnoise and only gives the correct result when no modulation sidebands are present, see Section 6.2.

- **ad** : amplitude detector
  usage : `ad name [n m] f node[*]`
  n, m = TEM mode numbers. n refers to the x-direction.
  f = sideband frequency in Hz (as offset to the input light)
  The optional asterisk behind the node name changes from the default beam to the second beam present at this node (see Section 3.2.1 for the definition of the default beam).
  The amplitude detector calculates field amplitude and phase of all light fields at **one** frequency. For correct absolute values you have to set 'epsilon_c' correctly in 'kat.ini' (see section 2.2). See the command '`yaxis`' for the various possibilities for plotting the computed values.

If higher order modes are used and no indices n, m are given, the amplitude detector tries to compute the value for the 'phase front' on the optical axis. See Section 4.8.1.

- **qd** : quantum quadrature detector
  usage : `qd name f phase node[*]`
  f = sideband frequency in Hz (as offset to the input light)
  phase = quadrature realtive to the amplitude quadrature (in degrees)
  The optional asterisk behind the node name changes from the default beam to the second beam present at this node (see Section 3.2.1 for the definition of the default beam).
  This outputs the amplitude of a quantum quadrature, e.g. amplitude or phase quadrature, for a specific carrier field. The frequency value given should be that of a carrier field or one of its modulated sidebands realtive to the set reference frequency of the simulation.
  The phase value determines which quadrature to measure. To output the amplitude quadrature use a phase 0, the phase quadrature is given using a phase of 90.
  The units of the output are normalised in units of $hf/2$, where h is Planck's constant and f is the simulation reference frequency. Thus an output of 1 means the quadrature is the same size of a pure vacuum state, less than 1 is squeezed and more than 1 means an increase in noise.
  Note that this is not finalised for use with higher-order modes yet.

- **sd** : squeezing detector
  usage : `sd name f [n m] node[*]`
  f = sideband frequency in Hz (as offset to the input light)
  n, m = TEM mode numbers. n refers to the x-direction.
  The optional asterisk behind the node name changes from the default beam to the second beam present at this node (see Section 3.2.1 for the definition of the default beam).
  This outputs information regarding the squeezing of the quantum noise state of a given carrier field. The frequency value given should be that of a carrier field or one of its modulated sidebands realtive to the set reference frequency of the simulation. You can also specify which optical mode to view too but this is not finalised for use with higher-order modes yet.
  The detector outputs a complex number whose magnitude is the squeezing level in dB r_{db}. The 'squeezing factor' r is related to r_{dB} by:

$$r_{db} = 20\, r\, \log_{10}(e). \tag{G.1}$$

  The phase of the complex output is ϕ the squeezing angle in degrees (use the `yaxis` command to output phase information). When $\phi = 0, 180$ and $r \neq 0$ then the carrier is purely amplitude squeezed, when $\phi = \pm 90$ and $r \neq 0$, the carrier is purely phase squeezed; any other angle is a mixture.

- **shot** : shot noise
  usage : `shot name node[*]`
  It calculates the shot noise in the output using the DC light power P as

$$\Delta P = \sqrt{\frac{2\,h\,c}{\lambda_0} P}\ . \tag{G.2}$$

  Note: This detector relies on a simple approximation for the shotnoise and only gives the correct result when no modulation sidebands are present, see Section 6.2.

- **qnoised** : photodiode quantum noise (plus one or more mixers)
  usage : `qnoised name n [f1 [phase1 [f2 [phase2 [...] ] ] ] ] node[*]`

n	=	number of demodulation frequencies ($0 \leq n \leq 5$)
f1	=	demodulation frequency of the first mixer in Hz
phase1	=	demodulation phase of the first mixer in degrees
f2	=	demodulation frequency of the second mixer in Hz
phase2	=	demodulation phase of the second mixer in degrees

  ...
  Will compute the photocurrent noise of a DC or demodulated photodiode output. Demodulations are set exactly the same as the `pd` commands. However the `max` phase keyword cannot be used and all demodulation phases must be specified.
  For radiation pressure and squeezing effects to be visible you must set the final demodulation frequency to the signal frequency, either using put commands or the `$fs` or `$mfs` variables.
  The scaling of the output is dependent on the `quantum_scaling` value in the `kat.ini` file. By default it should output amplitude spectral density noise with units of Watts/$\sqrt{Hz}$.
  qnoisedS can be used to output the quantum noise-to-signal ratio. It requires a signal injection via `fsig`. The output is given as: noise/signal where noise is computed using a `qnoise` detector and signal with a `pd[n]` detector. The same number of demodulations are applied to each detector. It is possible to use the `max` option as a phase with this command, as the phase needed to maximize the noise is compute first and applied to both noise and signal detectors.
  qnoiseN provides the reciprocal signal-to-noise output.

- **qshot** : photodiode shot noise (plus one or more mixers)
  usage : `qshot name n [f1 [phase1 [f2 [phase2 [...] ] ] ] ] node[*]`

n	=	number of demodulation frequencies ($0 \leq n \leq 5$)
f1	=	demodulation frequency of the first mixer in Hz
phase1	=	demodulation phase of the first mixer in degrees
f2	=	demodulation frequency of the second mixer in Hz
phase2	=	demodulation phase of the second mixer in degrees

  ...
  This will compute the demodulated noise present in the photocurrent output of a

photodiode. This detector assumes only that vacuum noise is present in the optical field. Thus it cannot see radiation pressure effects nor squeezing.
The scaling of the output is dependent on the `quantum_scaling` value in the `kat.ini` file. By default it should output amplitude spectral density noise with units of Watts/$\sqrt{Hz}$.
qshotS will output the quantum noise-to-signal ratio and **qshotN** the signal-to-noise ratio, see also `qnoised`.

- **pgaind** : motion parametric gain detector
  usage : `pgaind name component motion`
  name = name of the detector
  component = suspended component – mirror or beamsplitter
  motion = motion degree of freedom
  ...
  This detector can be used to compute the parametric gain of a particular motion of a suspended mirror or beamsplitter due to its coupling to the optical fields that may drive the motion. The value this detector outputs is:

$$\mathcal{R}_m = \mathcal{R}\rceil \left\{ \frac{\Delta A}{A} \right\} \tag{G.3}$$

  Where $\frac{\Delta A}{A}$ is the transfer function from a specified motion back into itself. Thus if $\mathcal{R}_m > 1$ the motion is unstable and will exponentially increase with time. With $0 < \mathcal{R}_m < 1$ the motion will build up but is not necessarily unstable, $\mathcal{R}_m < 0$ means the motion is being damped and will not resonate.
  The component specified must be suspended (for either longtudinal or rotational motions) or have a surface motion set. The possible motion values are then: `z` for longutindinal motion, `rx` or `yaw` for yaw motions, `ry` or `pitch` for pitch motions, and `sN` for surface motions, where `N` is the $(N-1)^{\text{th}}$ surface motion that is set using the `smotion` command.

- **mod** : modulator
  usage : `mod name f midx order am/pm [phase] node1 node2`
  f = modulation frequency in Hz
  midx = modulation index
  order = number of sidebands (or '`s`' for single sideband)
  am/pm = amplitude or phase modulation
  phase = phase of modulation
  Phase modulation :
  'order' sets the order up to which sidebands are produced by the modulator. For example, given an input light with 'f' equal to zero
  `mod mo1 10k 0.3 2 pm n1 n2`
  produces sidebands at -20kHz, -10kHz, 10kHz and 20kHz. The maximum possible order is 6. If order is set to '`s`' then the modulator produces a single sideband.

Amplitude modulation:
'order' is always set to 1, and `midx` must be between 0 and 1.

- **fsig** : signal frequency
  usage : `fsig name component [type] f phase [amp]`
  or
  usage : `fsig name f`
  or
  usage : `fsig name component [type] f tf_name`

component	=	one of the previously defined component names
type	=	type of modulation
f	=	signal frequency
phase	=	signal phase
amp	=	signal amplitude
tf_name	=	transfer function name

  Used as the input signal for calculating transfer functions, see also Section 3.1.2.
  For example
  `fsig sig1 m1 10k 0`
  shakes the previously defined component (e.g. a mirror) 'm1' at 10 kHz. Only one signal frequency can be used in one calculation but that can be fed to several components, e.g.
  `fsig sig1 m1 10k 0`
  `fsig sig2 m2 10k 0`
  inserts the signal at two mirrors (in phase).
  For the moment the following types of signal modulation are implemented (default type marked by *):
  - mirror: phase*, amplitude, xbeta, ybeta
  - beam splitter: phase*, amplitude, xbeta, ybeta
  - space: amplitude, phase*
  - input: amplitude, phase, frequency*
  - modulator: amplitude, phase*

  To tune only the signal frequencies one has to be explicitly tuned with `xaxis` or `put`, because only one signal frequency is allowed. E.g. in the example above, tuning `sig1` will also tune `sig2`.
  Alternatively `fsig` can be used to only specific a 'signal' or 'noise' frequency without injecting a signal. The syntax is:
  `fsig name f`
  This version is typically used when looking purely at quantum noise computations where no signals are required.
  Another alternative syntax for `fsig` is given as:
  `fsig name component [type] f tf_name`
  This is an idential command to the normal `fsig` input except that the phase and amplitude values are computed from a provided transfer function instead. See `tf`

and `tf2` commands for defining transfer functions.

- **vacuum** : Set which quantum noise inputs to use
  usage : `vacuum component1 component2 ...`
  component = name of component to include quantum noise input at
  This command is used to specify which components should be sources of quantum noise. By default Finesse will find any open port, lossy optic, squeezer or laser input and set it on to include all possible noise sources. If you instead wish to select only specific noise sources you can just specify a list of component names.
  Use the `printnoises` commmand to list all the noise sources FINESSE will use in the simulation.

G.3 Hermite-Gauss extension

This section gives the syntax of components and commands which are part of the Hermite-Gauss extension of FINESSE as described in Chapter 4. The command `maxtem` is used to switch between plane-waves and Hermite-Gauss beams, see below.

- **tem** : distribute input power to various TEM modes
  usage : `tem input n m factor phase`
  alternative : `tem* input p l factor phase`
  input = name of an input component
  n, m = Hermite − Gaussian$\mathrm{TEM_{nm}}$ mode numbers
  n, m = Laguerre − Gaussian$\mathrm{TEM_{pl}}$ mode numbers
  factor = relative power factor
  phase = relative phase in degrees
  When an input (component '`1`') is specified, the given laser power is assumed to be in the $\mathrm{TEM_{00}}$ mode. The command `tem` can change that. Each `tem` command can set a relative factor and phase for a given TEM mode at a specified input. Several commands for one input are allowed.
  Please note that the `tem` command is intended to *add* higher order modes, i.e.:
  `tem input1 0 1 1.0 0.0`
  adds a $\mathrm{TEM_{01}}$ mode to the $\mathrm{TEM_{00}}$ mode. Both fields have the same amplitude.
  In order to create a pure higher order mode the $\mathrm{TEM_{00}}$ amplitude has to be explicitly set to zero. For example:
  `tem input1 0 0 0.0 0.0`
  `tem input1 0 1 1.0 0.0`
  would put all power into the $\mathrm{TEM_{01}}$ mode.
  Another example:
  `tem input1 0 0 1.0 0.0`
  `tem input2 2 1 1.0 90.0`

specifies that exactly the same amount of power as in the TEM_{00} mode should be in TEM_{21}, but with a phase offset of 90 degrees.
You can also specify an input Laguerre-Gaussian beam using `tem*`. For this version of the command $p > 0$ but l can be a positive or negative integer.
Note: '`tem`' does not change the total power of the laser beam.

- **lens** : thin lens
  usage : `lens name f node1 node2`
  f = focal length in metres
  A lens does not change the amplitude or phase of a passing beam. When Hermite-Gauss modes are used, the beam parameter q is changed with respect to f.

- **bp** : beam parameter detector
  usage : `bp name x/y parameter node`
  x/y = direction for which the parameter should be taken
  parameter = a parameter derived from the Gaussian beam parameter, see below.
  This detector can plot a variety of parameters able to be derived from the Gaussian beam parameter which is set to the respective node:
  w : beam radius in metres
  w0 : waist radius in metres
  z : distance to waist in metres
  zr : Rayleigh range in metres
  g : Gouy phase in radians
  r : Radius of curvature (of phase front) in meters
  q : Gaussian beam parameter
  Please note that the Gouy phase as given by `bp` is not the accumulated Gouy phase up to that node but just the Gouy phase derived from the beam parameter is:

$$\Psi(z) = \arctan\left(\frac{\mathrm{Re}\,\{q\}}{\mathrm{Im}\,\{q\}}\right).$$

  The accumulated Gouy phase can be plotted with the detector `gouy`, see below.

- **cp** : cavity parameter detector
  usage : `cp cavity-name x/y parameter`
  cavity–name = name of cavity of which the parameter should be taken
  x/y = direction for which the parameter should be taken
  parameter = a parameter derived by the cavity trace algorithm, see below.
  Please note that this detector type has not a unique name. Instead the name of the respective cavity must be given.
  This detector can plot a variety of parameters that are derived by a cavity trace. The cavity must have been specified by the 'cav' command. Please note that these parameters are only filled with meaningful numbers when a cavity trace is executed. You can use the command 'retrace' to force a trace for each data point. The

available parameters are the same that can be printed to the terminal using 'trace 2':

w	:	beam radius at the first cavity node in metres
w0	:	waist radius at the first cavity node in metres
z	:	distance to waist at the first cavity node in metres
r	:	radius of curvature of beam phase front at the first cavity node in meters
q	:	Gaussian beam parameter at the first cavity node
finesse	:	cavity finesse
loss	:	round trip loss (0<=loss<=1)
length	:	optical path length of the cavity in meters (counting a full round-trip)
FSR	:	free spectral range in Hz
FWHM	:	cavity linewidth as Full Width at Half Maximum in Hz
pole	:	cavity pole frequency in Hz (=0.5*FWHM)
g	:	cavity stability parameter

The cavity stability ranges from -1 to 1 for stable cases. Please also note that the direction parameter (x/y) only applies to the parameters related to beam size and stability but must always be given so that the parsing of the 'cp' command will function.

- **gouy** : gouy phase detector
  usage : `gouy name x/y space-list`

x/y	=	direction for which the phase should be taken
space-list	=	a list of names of 'space' components

  This detector can plot the Gouy phase accumulated by a propagating beam. For example:
  `gouy g1 x s1 s2 s3 s4 sout`
  plots the Gouy phase that a beam accumulates on propagating through the components `s1`, `s2`, `s3`, `s4` and `sout`.

- **pdtype** : defines the type of a photodetector
  usage : `pdtype detector-name type-name`

  This command defines the type of a photodetector with respect to the detection of transverse modes. The standard detector is a simple photodiode with a surface larger than the beam width. With 'pdtype' more complex detectors can be used, like for example, split photodetectors.
  In the file 'kat.ini' a number of different types can be defined by giving scaling factors for the various beat signals between the different Hermite-Gauss modes. For example, if a photodetector will see the beat between the TEM_{00} and TEM_{01}, then the line '0 0 0 1 1.0' (mode factor) should be present in the description. The definitions in the 'kat.ini' file are given a name. This name can be used with the command 'pdtype' in the input files. Many different types of real detectors (like split detectors) or (spatially) imperfect detection can be simulated using this feature.
  The syntax for the type definitions:
  `PDTYPE name`

```
...
END
```

Between `PDTYPE` and `END` several lines of the following format can be given:

1. '`0 1 0 2 1.0`', beat between TEM_{01} and TEM_{02} is scaled with factor 1.0
2. '`0 0 * 0 1.0`', '*' means 'any': the beats of TEM_{00} with TEM_{00}, TEM_{10}, TEM_{20}, TEM_{30},... are scaled with 1.0
3. '`x y x y 1.0`', '`x`' or '`y`' also mean 'any' but here all instances of '`x`' are always the same number (same for '`y`'). So in this example all beats of a mode with itself are scaled by 1.0

Please note that **all beat signals which are not explicitly given are scaled with 0.0**. ('`debug 2`' somewhere in the input file will cause FINESSE to print all non-zero beat signal factors for all defined types.) Please take care when entering a definition, because the parser is very simple and cannot handle extra or missing spaces or extra characters.

The file 'kat.ini' in the FINESSE package includes the definitions for split photodetectors, see Section 4.8.3.

- **beam** : beam shape detector

  usage : `beam name [f] node[*]`

  f = frequency of the field component in Hz

  The optional asterisk behind the node name changes from the default beam to the second beam present at this node (see Section 3.2.1 for the definition of the default beam).

  With the beam analyser one can plot the cross-section of a beam, see Section 4.8.4. If no frequency is given the beam detector acts much like a CCD camera, it computes the light intensity per unit area as a function of x and y. The x-axis has to be set to either 'x' or 'y'. For example `xaxis beam1 x lin -10 10 100` sets the x-axis to tune the position in the x-direction from $-10w_0$ to $10w_0$ in 100 steps (for beam analyser 'beam1'). A second x-axis can be set to the perpendicular direction in order to plot the two dimensional cross-section of the beam. Thus the axes of the plot are scaled automatically by w_0, the waist size computed from the Gaussian beam parameter at the beam detector. The values for the waist size are printed to the terminal and given in the plot labels.

  If a frequency is given the beam detector outputs the amplitude and phase of the light field at the given frequency (you can use the `yaxis` command to define whether the amplitude, phase or both should be plotted).

- **mask** : mask out certain TEM modes in front of a photodetector or a beam analyser

  usage : `mask detector n m factor`

  detector = name of a photodetector or beam analyser

  n, m = TEM_{nm} mode numbers

  factor = power factor [0,1]

  Several `mask` commands can be used per detetector.

Without this command all photodetectors (for which 'pdtype' is not used) detect the power of all TEM modes, for example :

$$S(f_1, f_2) = \sum_{\mathrm{nm}} 2\,\mathrm{Re}\left(a_{\mathrm{nm}}(f_1) a_{\mathrm{nm}}(f_2)\right) = \sum_{\mathrm{nm}} S_{\mathrm{nm}}, \tag{G.4}$$

where $a_{\mathrm{nm}}(f)$ is the amplitude of the $\mathrm{TEM}_{\mathrm{nm}}$ mode at frequency f. Note that other detectors, like split detectors, quadrant cameras or bulls-eye detectors use a special geometry to detect certain cross-terms. Setting a mask for a $\mathrm{TEM}_{\mathrm{nm}}$ will scale the detected power by the given factor:

$$S_{\mathrm{nm}}(f_1, f_2) = \text{factor}\; 2\,\mathrm{Re}\left(a_{\mathrm{nm}}(f_1) a_{\mathrm{nm}}(f_2)\right). \tag{G.5}$$

- **attr** : additional (optional) attributes for mirrors, beam splitters and spaces
  usage : `attr component M value Rc[x/y] value x/ybeta value gx/y value`

component	=	the component for the attributes to be set to
M	=	mass in grammes
Rc	=	radius of curvature, in metres (zero is used for plane surface)
Rcx	=	radius of curvature in interferometer plane
Rcy	=	radius of curvature in plane perpendicular to interferometer
Rap	=	Radius of aperture (size of optic) [m] (Only for mirrors)
xbeta	=	angle of mis-alignment in the interferometer plane in radian
ybeta	=	angle of mis-alignment perpendicular to the interferometer plane in radian
g	=	Gouy phase of a space component in degrees (see Section 4.3.3)
gx	=	Gouy phase of a space component (horizontal component)
gy	=	Gouy phase of a space component (vertical component)
value	=	numerical value for the specified attribute

  Note, in contrast to phases **the alignment angles `xbeta` and `ybeta` are given in radians**.
  The various attributes are optional. For example one can simply set the radius of curvature of a mirror 'm1' to 10 metres with the command:
  `attr m1 Rc 10`
  The sign for the radius of curvature is defined as follows: if the surface seen from the **first specified node** (specified at the respective mirror or beam splitter) is concave, then the number for the radius of curvature is **positive** (see Section 4.3.4).
  Please note that when the attributes 'Rc' or 'g' are used you cannot tune the parameter itself. Instead, the separate directions i.e. 'Rcx' and/or 'Rcy' and 'gx' and/or 'gy' must be used for further tuning, e.g. with the `xaxis` command.

- **map** : loads and applies a surface map to a mirror or beamsplitter component
  usage : `map component filename`

component = mirror or beamsplitter to which the map should be applied
filename = name of the file containing the map data

This command reads a file given by filename and searches for a surface map given by a grid or by coupling coefficients (previously computed by FINESSE). The data must be provided in a special structure, see 4.7. You can apply multiple maps to a mirror surface by repeating the command with the same component.

- **smotion** : loads and applies a surface motion map to a mirror component

usage : `smotion component map_file transfer_function`

component = mirror to which the map should be applied
map_file = name of the file containing the map data
transfer_function = name of transfer function

This command reads a file given by map_file and searches for a surface motion map. This file describes the normalised motion of a surface for use in radiation pressure computations.
The data must be provided in a special structure, see 4.7. You can apply multiple surface motions to a mirror by repeating the command with the same component. The transfer function should describe how the optical force is converted into motion of this particular shape. FINESSE will internally handle the computation of the overlap between each optical mode and the surface motion shape.

- **gauss** : setting the q parameter for a Gaussian beam

usage : `gauss name component node w0 z [wy0 zy]`
(alternative: `gauss* name component node q [qy]`)
(alternative: `gauss** name component node w(z) Rc [wy(z) Rcy]`)

w0 = beam waist size in metres
z = distance to waist in metres
Rc = Radius of curvature
w(z) = beam spot size at node
q = complex beam parameter (given as '$z + \mathrm{i}z_\mathrm{R}$', i.e. 'distance-to-waist Rayleigh-range')

A Gaussian beam at a certain point z' on the optical axis can be characterised by two parameters. The first common method is to specify the waist size w_0 and the distance to the waist z. In FINESSE the complex parameter for Gaussian beams is used:

$$q = z + \mathrm{i}\, z_\mathrm{R},$$

with z_R the Rayleigh range and z the distance to the beam waist.
The distance to the waist can be positive or negative. **A positive value means the beam has passed the waist, a negative number specifies a beam moving towards the waist**. It is clear that the q parameter has to be set for a certain direction of propagation (the other direction then has the parameter $q' = -q^*$). The direction of propagation is set with 'component'. The node at which the Gauss parameter is to be set has to be connected to the specified component. The direction

of propagation is defined as: **from the component towards the node**.
In general a Gaussian beam may have two different beam parameters for the x- and the y-direction. When two parameter sets are given with `gauss` the first set is assumed to be valid for the x-direction and the second for the y-direction. If only one set is given then it is used for both directions.

- **cav** : tracing a beam through a cavity and compute the q-eigenvalues
  usage : `cav name component1 node1 component2 node2`
  The components and nodes specify the start and end point of a beam path through a possible cavity. '`node1`' has to be connected to '`component1`' and '`node2`' to '`component2`'.
  There are only two possibilities for specifying a cavity in FINESSE:
  - a linear cavity: the start component and end component are two different mirrors.
  - a ring cavity: the start component and end component are the same beam splitter and the nodes are either 1 and 2 or 3 and 4, so that the beams are connected to each other via a reflection.

  When the cavity is stable (not critical or unstable) the eigenmatrix is computed. The resulting eigenvalues for the Gaussian beam (q parameters) are then set for all cavity nodes.
  Use 'trace' in the input file to see what cavity nodes are found and which q-values are set, see below.

- **trace** : set verbosity for beam tracing
  usage : `trace n`
  n = an integer which sets the verbosity level.
  When the trace is set, FINESSE will print some information while tracing a beam through the interferometer, through a cavity, or during other computation tasks that are connected to the Hermite-Gauss extension. The integer 'n' is bit coded, i.e. n=2 gives different information to n=4, while n=6 will give both.

n	output
1	list of TEM modes used
2	cavity eigenvalues and cavity parameters like free spectral range, optical length and finesse
4	mode mismatch parameters for the initial setup (mismatch parameter as in [Bayer-Helms])
8	beam parameters for every node, nodes are listed in the order found by the tracing algorithm
16	Gouy phases for all spaces
32	coupling coefficients for all components
64	mode matching parameters during calculation, if they change due to a parameter change, for example by changing a radius of curvature
128	nodes found during the cavity tracing

- **retrace [off|force]** : recomputes the Gaussian parameters at each node for every data point
  usage : `retrace`
  FINESSE needs to trace the beam through the interferometer in order to set the Gaussian beam parameters (see Section 4.4 for a detailed description). This is always done once at the start of a simulation if higher-order modes are used. If `xaxis` or `put` are used to tune a parameter like a length of a space or the focal length of a lens or the radius of curvature of a mirror the beam parameters are locally changed and the beam tracing should be repeated. Without re-computing a proper base of Gaussian beam parameters such a tuning introduces a virtual mode mismatch which can lead to wrong results.
  FINESSE automatically detects whether a re-tracing is required and if so computes a new set of base parameters for each data point. The `retrace` command can be used to over-ride the automatic behaviour: `retrace` will force a retracing and `retrace off` will prevent it, regardless of the `xaxis` settings.
  Please note that the re-tracing cannot avoiding all unwanted mode-mismatches.
  Using 'retrace force' will force the tracing alogorithm to continue tracing even if it comes across an unstable cavity.

- **startnode** : recomputes the Gaussian parameters at each node for every data point
  usage : `startnode node`
  This command allows one to explicitly set the node at which the automatic beam trace algorithm starts. The node must have a beam parameter associated with it. This means the node must be inside a cavity that has been traced with the `cav` command or the parameter must be explicitly set via the `gauss` command.

- **maxtem** : set the maximum order for Hermite-Gauss modes
  usage : `maxtem order`
  order = maximum order, i.e. $n + m$ for $\mathrm{TEM}_{\mathrm{nm}}$ modes.
  maximum number : 1

  This defines the maximum order for $\mathrm{TEM}_{\mathrm{nm}}$ and thus the number of light fields used in the calculation. The default 'order' is 0, the maximum value is 100. For large values the interferometer matrix becomes very large and thus the simulation extremely slow. Please note that the Hermite-Gauss mode is automatically switched on if at least one attribute or command referring to transverse modes is entered. You can explicitly switch off the Hermite-Gauss mode by using 'maxtem off'.

- **phase** : switches between different modes for computing the light phase
  usage : `phase number`

Four different modes are available:
0 = no change
1 = the phase for coupling coefficients of TEM_{00} is scaled to 0
2 = the Gouy phase for TEM_{00} is scaled to 0
3 = combines modes 1 and 2 (default)
The command 'phase' can be used to change the computation of light field phases in the Hermite-Gauss mode. In general, with higher order modes the spaces are not resonant any more for the TEM_{00} mode because of the Gouy phase. Furthermore, the coupling coefficients k_{nmnm} contribute to the phase when there is a mode mismatch. For correct analysis these effects have to be taken into account. On the other hand, these extra phase offsets make it very difficult to set a resonance condition or operating point intuitively. In most cases another phase offset would be added to all modes so that the phase of the TEM_{00} becomes zero. With the command 'phase' these phase offsets can be set for the propagation through free space, for the coupling coefficients or both: 'phase 1' ensures that phases for the coupling coefficients k_{0000} (TEM_{00} to TEM_{00}) are 0, 'phase 2' ensures that all Gouy phases for TEM_{00} are 0 and 'phase 3' combines both effects. The phases for all higher modes are changed accordingly, so that the relative phases remain correct. **Please note that only `phase 0` and `phase 2` guarantee always correct results, see Section 4.10.2 for more details.**

- **knm** : specifies a file which the coupling coefficients for a given component generate and saves/loads them from the file
  usage : `knm component_name filename_prefix`

  A different command of the same name existed in previous versions. Usage has changed from version 0.99.8. The old functionality has been moved into the `conf knm_flags` command.
  With 'knm' the user can specify a file which the coupling coefficents for a component with a map applied can save and load the coefficients from. This is primarily used to save computational time as map coefficients can be expensive to compute. The only components this can be applied to at the moment are mirrors. `filename_prefix` states the filename prefix.
  The numeric integration uses a fast self-adapting routine [DCUHRE] and Cuba routines but nevertheless will be very slow in comparison to the simple formula calculation. The numeric integration algorithm can be customised with the following parameters in the kat.ini file:

maxintop	:	maximum function calls of the numeric integration algorithm (default 400000)
abserr	:	absolute error requested by integration routine (default 1e-6)
relerr	:	relative error requested by integration routine (default 1e-6)
maxintcuba	:	Maximum integrand calculations for Cuba routines (default 1e6)

G.4 Commands

- **xaxis** : x-axis definition, i.e. parameter to tune
  usage : `xaxis component parameter lin / log min max steps`
  (alternative: `xaxis* component parameter lin / log min max steps`)

component	=	one of the previously defined component names
parameter	=	a parameter of the component e.g. 'L' for a space
lin/log	=	defines a linear or logarithmic x-axis
min	=	start value
max	=	stop value
steps	=	number of steps from min to max

  maximum number : 1

  The previous definition of the interferometer yields exactly one output value for every detector. To create a plot we have to define a parameter that is changed (tuned/swept) along the x-axis of the plot. Exactly one `xaxis` must be defined. For example,
  ```
  xaxis s1 L lin 1 10 100
  ```
  changes the length of space s1 from 1 metre to 10 metres in 100 steps.
  Another useful example is to sweep the laser frequency using:
  ```
  xaxis i1 f lin 0 10k 500
  ```
  When the optional asterisk is used then the previously defined value for the parameter to tune is used as an offset. For example:
  ```
  s s1 L 5
  xaxis* s1 L lin 1 10 100
  ```
  tunes the length of space `s1` from 6 to 15 metres.
  When the axis is logarithmic the min/max values are multiplied to the previously defined value, e.g.
  ```
  s s1 L 5
  xaxis* s1 L log .1 10 100
  ```
  tunes the length of space `s1` from 0.5 to 50 metres. Note that the parameters used as x-axis in the output plot are those given in the `xaxis*` statement, **not** the computed values which are really used in the calculation. This feature allows one to specify

the tunings of the operating point in the interferometer description and then always tune *around* that operating point by min = −max.

- **x2axis** : second x-axis definition, creates 3D plot
  usage as for `xaxis`

  This command defines a second x-axis. A 3D plot is created.

- **yaxis** : y-axis definition (optional)
  usage : `yaxis [lin / log] abs:deg / db:deg / re:im / abs / db / de`

abs:deg	=	amplitude and phase
db:deg	=	amplitude in dB and phase
re:im	=	real and imaginary part
re	=	real part
im	=	imaginary part
abs	=	amplitude
db	=	amplitude in dB
deg	=	phase

  maximum number : 1

  This defines the (first plus an optional second) y-axis.

- **scale** : rescaling of output amplitudes (optional)
  usage : `scale factor [detector]`

factor	=	scale factor
detector	=	output name

  All or a specified output signal is scaled by factor. (The scaling is done after demodulations.)
  If the keyword `meter` is used instead of a number for `factor` the output is scaled by $2\pi/\lambda_0$ (or $\lambda_0/2\pi$ for `pdS`). In case the output is a transfer function and the signals have been added to mirrors or beam splitters the transfer functions are thus normalised to W/m/$\sqrt{\text{Hz}}$ (m/$\sqrt{\text{Hz}}$ for `pdS`).
  If the keyword `ampere` is used instead of a number for `factor` the output is scaled by $e\ q_{\text{eff}}\ \lambda_0/(hc)$. This converts light power (Watt) to photocurrent (Ampere).
  If the keyword `deg` is used the output will be scaled by $180/\pi$.

- **diff** : differentiation
  usage : `diff component parameter`

component	=	one of the previously defined component names
parameter	=	a parameter of the component e.g. 'L' for a space

  maximum number : 3

Instead of the standard result of the calculation, partial differentiation with respect to the specified parameter will be plotted. For a higher order differentiation you can specify the command again with the same parameter. The differentiation is calculated as:
`diff = (f(x+h/2) - f(x-h/2))/h`
The step size `h` can be specified via the constant `deriv_h` in 'kat.ini', the default is 1e-3. You can also overwrite the value from the 'kat.ini' file with the command `deriv_h` in an input file. This is useful when several files which require a different step size are located in the same directory.
Please note that if `put` is used, the parameter specified in `put` is linked to the parameter from the `xaxis` command, so a differentiation with respect to the parameter specified in `put` is not possible and a differentiation with respect to the parameter stated in `xaxis` will automatically perform a differentiation with respect to the connection of parameters (which was introduced by `put`).

- **const** : constant definition
  usage : `const name value`
  name = user-defined name, less than 15 characters long
  value = numerical or string value
  The constants are used during pre-processing of the input file. If anywhere in the file the command '`const name value`' is defined, every instance of '`$name`' in the input file is replaced by '`value`'.

- **variable** : definition of a dummy variable
  usage : `variable name value`
  name = user-defined name, less than 15 characters long
  value = numerical or string value
  The sole purpose of this command is to provide a dummy variable that can then be tuned by the `xaxis` command and connected to the interferometer with `put` commands. A typical application would be the tuning of a differential arm length as follows:

```
variable deltax 1
xaxis deltax abs lin -1 1 100
put* ETMX phi $x1
put* ITMX phi $mx1
```

- **set** : variable definition
  usage : `set name component parameter`
  name = user defined name, less than 15 characters long
  component = one of the previously defined component names
  parameter = a parameter of the component e.g. 'L' for a space
  The variables are used for creating input variables that can be used with functions, see below.

With 'set', all tunable parameters in the input file can be accessed with the usual syntax '`component parameter`'. The `set` command will link the variable '`$name`' to the parameter value of the named component. In addition, the output of any detector can be stored in a variable. The syntax is:
`set name detector-name re/im/abs/deg`
where `re/im/abs/deg` indicate which real number to use if the detector output is a complex number. (NOTE: for detectors with a real output, use '`re`' and NOT '`abs`' since '`abs`' will remove the sign!)
The `set` commands are executed for each data point, i.e. if the component parameter is changed e.g. with the `xaxis command` the variable `$name` will change accordingly.
In addition to user defined variables, several internal variables have been defined: '`$x1`', '`$x2`' and '`$x3`' have been pre-defined and point to the current value of the `xaxis` (or `x2axis`, `x3axis` respectively). in addition, '`$mx1`', '`$mx2`' and '`$mx3`' are defined as minus the corresponding '`$x`' variable. These predefined names must not be used with '`set`'.

- **func** : function definition
  usage : `func name = function-string`
  name = user-defined name, less than 15 characters long
  function-string = a mathematical expression
  For example, `func y = $Lp+2` defines the new variable 'y = Lp+2', with 'Lp' being a previously defined variable. Such previously defined variables are entered with a '$' sign. The new variable (i.e. the function result) will be plotted as a new output, like a detector output. Any previously defined variable via `set`, `func` or `lock` (see below) can be used like the function string. The functions are exectuted for each data point. (Please note that if you use two similar function names like 'function' and 'function1' the parser might have problems to distinguish between the two.)
  Note also that each function will be called initially when other variables and functions are probably not yet initialised. This can lead to a division-by-zero error. For example, the following line will often fail:

  ```
  func myfunc = $var1 / $func2
  ```
  To mitigate this you can use the following trick:
  ```
  func myfunc = $var1 / ( $func2 + 1E-21 )
  ```

  This feature uses the mathematical expression parser Formulc 2.22 by Harald Helfgott. The following functions are available in the function string: exp(), ln(), sin(), cos(), tan(), asin(), acos(), atan(), atan2(), abs(), sqrt(), pi() (with no parameters inside the parentheses) and rnd() (a random number between 0 and 1).
  Numbers have to be given numerically, e.g. '3.0E-9' instead of '3n'. Please note that '3.0e-9' does not work. Multiplication with negative numbers requires parentheses, e.g.:
  ```
  y = (-1)*$x1
  ```

For a detailed description of the parser syntax, please see the documentation of Formulc 2.22.

- **put[*]** : write variable into interferometer parameter
  usage : `put component parameter $variable`
  component = one of the previously defined component names
  parameter = a parameter of the component e.g. 'L' for a space
  variable = previously declared variable
  For example, `put space2 L $y` writes the content of variable y into the length of '`space2`'. (`put*` always adds to the initially set lengths of '`space2`')
  All `put` commands are executed once before first data point is computed. If photodetector outputs are used in `put` or `func` they are set to 0.0 for the first data point calculation.

- **tf** : pole-zero transfer function
  usage : `tf name factor phase [p or z f1 Q1 [p or z f2 Q2 ...]]`
  name = user-defined name, less than 15 characters long
  factor = overall scaling factor
  phase = overall phase factor
  p *or* z = add a pole or zero
  f1,f2,f3,... = frequency of pole or zero in Hz
  Q1,Q2,Q3,... = quality factor of pole or zero, $Q > 0$
  This command defines a generic transfer function with respect to frequency. Multiple poles and zeros can be defined at user-defined frequencies and quality factors for the resonances. For example:
  ```
  tf sus 1 0 p 1 100000 z 10 130 p 20 10M
  ```

- **tf2** : pole-zero transfer function
  usage : `tf2 name factor phase [p1,p2,...] [z1,z2,...]`
  name = user-defined name, less than 15 characters long
  factor = overall scaling factor
  phase = overall phase factor
  p1,p2,p3,... = complex pole
  z1,z2,z3,... = complex zero
  This command defines a generic transfer function. Rather than defining the transfer function with set frequencies and quality factors this accepts raw complex values. Each complex value should have a corresponding conjugate value defined. For example:
  ```
  tf2 pendulum 1 0 {1+100i,1-100i} {-3+10i,-3-10i}
  ```

- **lock** : control loop definition
  usage : `lock name $variable gain accuracy`

name	=	user defined name, less than 15 characters long
variable	=	previously defined variable (usually a photodetector output)
gain	=	loop gain
accuracy	=	threshold to decide whether the loop is locked

The command will read the variable given by '`$variable`' and write it into the new variable '`name`'. This variable will be also plotted as a new output, like a detector output. `lock*` stops after the first point so that only the initial lock is found and the rest is computed without locking. (Please note that if you chose two similar lock names like '`mylock`' and '`mylock1`' the parser might have problems to distinguish between the two.)
FINESSE will perform an iterative computation for each data point on the x-axis. In fact, it will compute the interferometer iteratively until the condition

$$|(\$\text{variable})| < \text{accuracy}$$

is fulfilled.
In order to achieve this goal the command tries to mimic a control loop with a simple integrator. The input '`$variable`' serves as the error signal and the output stored in '`$name`' holds the feedback signal (which has to be connected to the interferometer by the user with a `put` command). In each iterative step it perfroms the operation:

name = $name + gain * $variable (or name += gain * $variable)

Several `lock` commands can be active simultaneously and the lock output variables can be used in `func` commands located below the `lock` in the input file. **Please note: The order of the commands '`func`' and '`lock`' in the input file determines the order of their computation!**
Of course the lock fails miserably if:

- the loop is not closed,
- the error signal is not good,
- the computation is not started at or close to a good operating point,
- the gain is wrong (sign, amplitude) or,
- the steps as given by the `xaxis` command are too large (i.e. move the interferometer out of the linear range of the error signal).

A fine tuning of the gain is useful to minimise the computation time.
An example:

```
to lock a cavity to a laser beam we can write:
# laser and EOM
l i1 1 0 n0
mod eo1 40k 0.3 3 pm n0 n1
# cavity:
m m1 0.9 0.1 0 n1 n2
s s1 1200 n2 n3
m m2 .9 0.01 0 n3 n4
# Pound-Drever-Hall signal
pd1 pdh 40k 0 n1
```

```
# tune
xaxis m2 phi lin 0 100 400

# set the error signal to be photodiode output ('re' stands
# for the real part of the diode output. 're' has to be used
# with diodes that provide a real output.
set err pdh re
# Perform the lock! Gain is -10 and the accuracy 10n ( = 1e-8)
lock z $err -10 10n
# ... and connect the feedback to the interferometer
put* m1 phi $z
```

The behaviour of the locking routine can be adjusted by setting some paramaters in 'kat.ini'. For example, the lock iteration can automatically adjust the loop gains. The following parameters in the 'kat.ini' file can be used:

- `locksteps` (integer, >0, default 10000): maximum number of steps in which the iteration tries to achieve the lock.
- `autogain` (integer, 0,1,2 default 2): switch for the automatic gain control: 0 = Off, 1 = On, 2 = On with verbose output.
- `autostop` (integer, 0,1 default 1): if autostop is swiched ON the locking algorithm will stop after it fails to reach the desired accuracy once.
- `sequential` (integer, 0,1,5, default 5): this keyword determines if the feedback signals are computed sequentially or in parallel. The sequential mode is slower but performs much better far away from the operating point or when 'autogain' is needed. The default 5 uses the sequential mode for the first two data points and then switches to the faster parallel locking.
- `lockthresholdhigh` (double, >0, default=1.5): whether or not a loop is probably oscillating with a too high gain is determined using 'lockthresholdhigh'. The criterion used is as follows (with y1,y2,... as successive error signal values): the oscillation condition is defined as:
  if abs((y1+y3-2*y2)/accuracy/y3) > lockthresholdhigh, true=loop oscillates.
- `lockthresholdlow` (double, >0, default=0.01): whether or not a loop gain is too low is determined using 'lockthresholdlow'. The low-gain condition is defined as:
  if abs((y1+y3-2*y2)/accuracy/y3) < lockthresholdlow, true=loop gain too low.
- `locktest1` (integer, >0, default 5) and locktest2 (integer, >0, default 40): 'locktest1' and 'locktest2' determine the number of steps that an iteration is allowed to remain in an 'oscillation' (or 'low gain'). After 'locktest1' number of steps the loop state is checked. If for 'locktest2' number of checks the same error condition persists the loop gain will be reduced or increased by the factor 'gainfactor'.
- `gainfactor` (double, >0, default 3).

You can find two more examples in Section 3.5 and a more detailed tutorial in Appendix A.

- **showiterate** : define verbosity of the `lock` commands
  usage : `showiterate steps`
  steps = number of iterations
  If 'steps' is >0 the current state of the lock iteration is printed every 'steps' iterations. If 'steps'= −1 the result is printed only after the first succesful iteration (useful for knowing the values of the initial operating point).

- **printfrequency** : print all frequencies use
  usage : `printfrequency`

  This will print a table listing all the frequencies used in the simulation: carriers, modulation sidebands and signal/quantum sidebands.

- **printnoises** : print all quantum noises used
  usage : `printnoises`

  This will print a list of all the quantum noise sources being considered in the simulation and at which components and nodes it is present.

- **lambda** : set default wavelength/frequency
  usage : `lambda wavelength`
  wavelength = new reference wavelength in meters
  With this command you set a new reference wavelength, overwriting that set in the kat.ini file (typically 1064nm).

- **noplot** : suppress the plot of an output
  usage : `noplot output`
  output = previously defined output (detector, function, etc.)
  Since `func` and `lock` create new outputs, the resulting plots might become very cluttered. Therefore the command `noplot output` has been introduced. It suppresses the plotting of the given output (photodetector, function, lock, ...). The data is stored in the *.out file as before, only the plot command in the respective the *.gnu batch file is changed.
  Please note that `noplot` cannot be used to suppress all plotting. One output must remain to be plotted. If you want to suppress all graphical output please use `gnuterm no`.

- **deriv_h** : overwrites the value for deriv_h given in 'kat.ini'
  usage : `deriv_h value`
  value = step size for numerical differentiation

This command can be used to overwrite the pre-defined vale for deriv_h. This can be useful especially when you want to differentiate alignment signals in which numerical values of 10^{-9} are often required.

- **conf** : Sets configuration options for various optical components
  usage : `conf component option value`
  component = mirror or beamsplitter name
  option = name of option to set
  value = input for option
  This command is used to fine tune and alter the computational routines for a given optical component. It does not represent anything physical about the optic. Currently it is used to alter the behaviour of the coupling coefficient computation for both mirrors and beamsplitters, it is not used for any other components at the moment.
  - `integration_method` (1 or 2 or 3) sets the numerical integration method. Cuba refers to a self-adapting routine which is faster but less robust: 1 - Riemann Sum, 2 - Cubature - Serial, 3 - Cubature - Parallel (default)
  - `interpolation_method` (1 or 2 or 3) set the interpolation method for the numerical integration of surface maps, the (use NN for maps with sharp edges): 1 - Nearest Neighbour (default), 2 - Linear, 3 - Spline.
  - `interpolation_size integer` (odd integer > 0) sets the size of the interpolation kernel, must be odd and > 0
  - `knm_flags` (integer > 0) Sets the knm computation flags which define if coeffs are calculated numerically or analytically if possible.
  - `show_knm_neval` (0/1) Shows the number of integrand evaluations used for the map integration.
  - `save_knm_matrices` (0/1) If true the knm matrices are saved to .mat files for distortion, merged map and the final result
  - `save_knm_binary` (0/1) If true the knm and merged map data is stored in a binary format rather than ascii. See -convert option for kat in -h for converting between the 2 formats
  - `save_interp_file` (0/1) If 1 then for each knm calculated a file is written to outputting each interpolated point. The output file will have 4 columns: x,y,A,phi. So for each integrand evaluation the interpolated point is plotted
  - `save_integration_points` (0/1) If 1 then the points used for integration are saved to files. Only use this with Riemann integrator, Cuba can use millions of points and is slow
  - `knm_order` (12 or 21) changes order in which the coupling coefficient matrices are computed. 1 = Map, 2 = Bayer-Helms
  - `knm_change_q` (1 or 2) Decides the value of the expansion beam parameter q_L. If 1 then q_L = q'_1 and if 2 then q_L = q_2.

G.5 Auxiliary plot commands

- **gnuterm** : Gnuplot terminal (optional)
  usage : `gnuterm terminal [filename]`
  terminal = one terminal name specified in 'kat.ini', default is 'x11' or 'windows' respectively
  filename = name for Gnuplot output file
  maximum number : 20

  If you do not want a Gnuplot batch file to be written use : 'gnuterm no'.

- **pause** : pauses after plotting
  usage : `pause`

  maximum number : 1

  Adds a command 'pause -1' to the Gnuplot batch file after each plot into a screen terminal.

- **multi** : switches from a single surface to multiple surfaces in 3D plots
  usage : `multi`

  maximum number : 1

  By default in a 3D plot only the first output is plotted even if multiple outputs are present. If 'multi' is set, Gnuplot plots multiple surfaces into the same graph.
  Please note that even without setting 'multi' the data of all outputs is present in the output data file.

- **GNUPLOT ... END** : extra Gnuplot commands
  usage (for example):

```
GNUPLOT
set view 70, 220, ,
set contour
END
```

  All the Gnuplot commands specified between GNUPLOT and END will be written to the Gnuplot batch file. This is especially useful for 3D plots (see 3D.kat for an example).

Acknowledgements

Gerhard Heinzel has been the major force behind the creation of FINESSE. He had the idea of using the LISO routines on interferometer problems and he let me copy his code for that purpose. Furthermore he has been very busy as my most faithful beta tester and has helped me getting the right ideas in many discussions. I have gotten many helpful bug reports from Guido Müller early on and always enjoyed discussing interferometer configurations with him. The latter is also true for Roland Schilling: For hours he would listen to me on the phone while I was trying to understand my program - or interferometers and optics. Ken Strain has been a constant source of help and support during the several years of development. During my time at Virgo Gabriele Vajente and Maddalena Mantovani have acted as faithful test pilots for the extension with the `lock` command. Alexander Bunkowski has initiated and helped debugging the grating implementation. Jerome Degallaix has often helped with suggestions, examples and test results based on his code OSCAR to further develop and test FINESSE.

Paul Cochrane has made a big difference with his help on transforming the source code from its messy original form into a more professional package, including a testsuite, an API documentation and above all a readable source code.

More recently (2011 to 2013) several current and former members from my research group in Birmingham have put significant effort into the further development, testing and use of FINESSE. Daniel Brown became lead programmer and provided a large number of bug fixes and new features navigating the tricky grounds of optics with high-order modes. Due to his work, FINESSE has reached version 1.0 and is now available as open source. Charlotte Bond is a specialist in using FINESSE, in particular with mirror surface maps or strange beam shapes; she has become the main contributor of our Simtools package and her help with FINESSE has been invaluable for getting the physics of higher-order modes right. Further, Keiko Kokeyama, Paul Fulda and Ludovico Carbone have worked very hard to help making FINESSE do useful things for the Advanced LIGO commissioning team.

In the last year Daniel Brown has implemented the long requested feature of radiation pressure effects in addition to a full quantum noise treatment in the two-photon formalism. Daniel was supported in this activity by other members of my group, especially Mengyao Wang and Rebecca Palmer; and we once again could rely on crucial assistance from Jan Harms.

Many people in the gravitational wave community have helped me with feedback, bug reports and encouragement. Some of them are Seiji Kawamura, Simon Chelkowski, Keita Kawabe, Osamu Miyakawa, Rainer Künnemeyer, Uta Weiland, Michaela Malec, Oliver Jennrich, James Mason, Julien Marque, Mirko Prijatelj, Jan Harms, Oliver Bock, Kentaro Somiya, Antonio Chiummo, Holger Wittel, Hartmut Grote, Bryan Barr, Sabina Huttner, Haixing Miao, Benjamin Jacobs, Stefan Ballmer, Nicolas Smith-Lefebvre, Daniel Shaddock and probably many more that I have not mentioned here.

Last but not least I would like to thank the GEO 600 group, especially Karsten Danzmann

and Benno Willke, for the possibility to work on FINESSE in parallel to my experimental work on the GEO site. FINESSE would not exist without their positive and open attitude towards the young members of the group.

Bibliography

[Abramowitz] Abramowitz M and Stegun I. A 'Handbook of mathematical functions with formulas, graphs, and mathematical tables', Dover Books, New York, 1965. 104

[Ballmer] S. Ballmer, J. Degallaix, A. Freise and P. Fulda: 'Comparing Finesse simulations, analytical solutions and OSCAR simulations of Fabry-Perot alignment signals', LIGO note T1300345, https://dcc.ligo.org/LIGO-T1300345, (2013). 185

[Bayer-Helms] F. Bayer-Helms: 'Coupling coefficients of an incident wave and the modes of a spherical optical resonator in the case of mismatching' (and references within), Appl. Opt. 23 (1984) 1369–1380. 82, 83, 106, 210

[Bond13] C. Bond, D. Brown and A. Freise: 'Interferometer responses to gravitational waves: Comparing Finesse simulations and analytical solutions', LIGO note T1300190, https://dcc.ligo.org/LIGO-T1300190, and arXiv.1306.6752, http://arxiv.org/abs/1306.6752, (2013). 185

[Bond13b] C. Bond, P. Fulda, D. Brown and A. Freise: 'Comparing Simulations of the Advanced LIGO Dual-Recycled Michelson', LIGO note T1400270, https://dcc.ligo.org/LIGO-T1400270, (2013). 185

[Bond13c] C. Bond, P. Fulda, D. Brown and A. Freise: 'Investigation of beam clipping in the Power Recycling Cavity of Advanced LIGO using Finesse', LIGO note T1300954, https://dcc.ligo.org/LIGO-T1300954, (2013). 185

[Bond14] C. Bond, P. Fulda, A. Freise and D. Brown: 'Simulations of effects of LLO mode-mismatches on PRFPMI error signals', LIGO note T1400182, https://dcc.ligo.org/LIGO-T1400182, (2014). 185

[Brown] D. Brown et al., paper in preparation. 122, 131

[Bunkowski01] A. Bunkowski, O. Burmeister, K. Danzmann and R. Schnabel: 'Input-output relations for a three-port grating coupled Fabry-Perot cavity', Opt. Lett. 30 (2005) 1183–1185. 46, 49

[Caves] C. M. Caves and B. L. Schumaker: 'New formalism for two-photon quantum optics. I - Quadrature phases and squeezed states. II - Mathematical foundation and compact notation'. Physical Review A, 31, 3068-3111 (1985). 131

[Clarke] J. Clarke, H. Wang, D. Brown and A. Freise: 'Revisiting Sidebands of Sidebands in Finesse', LIGO note T1300986, https://dcc.ligo.org/LIGO-T1300986, (2013). 185

[DCUHRE] J. Berntsen, T. O. Espelid and A. Genz: 'Algorithm 698; DCUHRE: an adaptive multidemensional integration routine for a vector of integrals', ACM Transactions on Mathematical Software (TOMS), 17, 4 (1991) 452–456. 212

[Dooley] K. Dooley et. al., 'Report from the Commissioning Workshop at LLO (Jan. 2013), LIGO note T1300497, https://dcc.ligo.org/LIGO-T1300497, (2013). 185

[Evans] M. Evans, L. Barsotti and P. Fritschel: 'A general approach to optomechanical parametric instabilities', Physics Letters A , 2010, 374, 665 - 671. 127, 128, 129

[FINESSE] A. Freise: 'FINESSE, Frequency domain interferometer simulation software (1999-2013), http://www.gwoptics.org/finesse/.

[Freise] A. Freise: 'The Next Generation of Interferometry: Multi-Frequency Optical Modelling, Control Concepts and Implementation', Ph.D. Thesis, University of Hannover (2003), http://www.amps.uni-hannover.de/dissertationen/freise_diss.pdf. 3

[Freise03] A. Freise, G, Heinzel, H. Lück, R. Schilling, B. Willke and K. Danzmann: 'Frequency-domain interferometer simulation with higher-order spatial modes', Class. Quantum Grav. 21 1067–1074 (2003). 3

[Freise10] A. Freise and K. Strain: 'Interferometer Techniques for Gravitational-Wave Detection', Living Reviews in Relativity, 2010, 13, 1-+, available at http://relativity.livingreviews.org/Articles/lrr-010-1/2. 5

[GEO] K. Danzmann et al.: in *First Edoardo Amaldi Conference on Gravitational Wave Experiments*, Frascati (1994), (World Scientific, Singapore, 1995) 100–111. 3

[Gnuplot] T. Williams, C. Kelley et.al.: Gnuplot (1999), http://www.gnuplot.info. 2, 4

[Grote] H. Grote, A. Freise, M. Malec, G. Heinzel, B. Willke, H. Lück, K. A. Strain, J. Hough and K. Danzmann: 'Dual recycling for GEO 600', Class. Quantum Grav. 21 473–480 (2003). 3

[GWIC] Numerical modelling tools for gravitational wave detectors, hosted by the Gravitational Wave International Committee (GWIC), https://gwic.ligo.org/simulations. 3

[gwoptics/impact] History and impact of the FINESSE simulation tool http://www.gwoptics.org/finesse/impact.php. 3

[H1kat] C. Bond, P. Fulda, A. Freise, L. Carbone, K. Kokeyama and D. Brown: 'Finesse input files for the H1 interferometer', LIGO note T1300904, https://dcc.ligo.org/LIGO-T1300904, (2013). 185

[Harms] J. Harms, P. Cochrane and A. reise: 'Quantum-noise power spectrum of fields with discrete classical components', Phys. Rev. A, 76, 023803-+ (2007), also available at http://arxiv.org/abs/quant-ph/0703119. 134, 170

[Heinzel] G. Heinzel: 'Advanced optical techniques for laser-interferometric gravitational-wave detectors',Ph.D. Thesis, University of Hannover (1999). 32, 33, 37

[Hewitson04] K. Hewitson: 'On aspects of characterising and calibrating the interferometric gravitational wave detector, GEO 600', PhD Thesis, University of Glasgow (2004), available at www.astro.gla.ac.uk/users/hewitson/MRHthesis.pdf. 171

[Hello-Vinet] P. Hello and J. Y. Vinet: 'Ënalytical models of thermal aberrations in massive mirrors heated by high power laser beams' J. Phys. France 51 1267 (1990). 178

[Kawabe] K. Kawabe: 'An informal note on FINESSE: "Sidebands of Sidebands"', internal note, available on the FINESSE webpage (2002).

[KLU] Davis T. A: 'CSparse, CXSparse, KLU, and BTF: Direct Methods for Sparse Linear Systems', SIAM, Philadelphia (2006).

[Kokeyama] K. Kokeyama, K. Arai, P. Fulda, S. Doravari, L. Carbone, D. Brown, C. Bond and A. Freise, 'Finesse simulation for the alignment control signal of the aLIGO input mode cleaner', LIGO note T1300074, https://dcc.ligo.org/LIGO-T1300074, (2013). 185

[Kimble] H. Kimble, Y. Levin, A. Matsko, K. Thorne and S. Vyatchanin, Conversion of conventional gravitational-wave interferometers into quantum nondemolition interferometers by modifying their input and/or output optics, Physical Review D, 2002 142

[L1kat] C. Bond, P. Fulda, A. Freise, L. Carbone, K. Kokeyama and D. Brown: 'Finesse input files for the L1 interferometer', LIGO note T1300901, https://dcc.ligo.org/LIGO-T1300901, (2013). 185

[LISO] G. Heinzel : 'LISO, Program for Linear Simulation and Optimization of analog electronic circuits', MPQ Garching (1998). 3

[LUXOR] J. Harms : 'LUXOR, graphical user interface for FINESSE', http://www.gwoptics.org/finesse/luxor.php 4, 5

[Lück] H. Lück, A. Freise, S. Goßler, S. Hild, K. Kawabe and K. Danzmann : 'Thermal correction of the radii of curvature of mirrors for GEO 600', Class. Quantum Grav. 21 985–989 (2003). 3

[Malec] M. Malec, H. Grote, A. Freise, G. Heinzel, K. A. Strain, J. Hough and K. Danzmann : 'Towards dual recycling with the aid of time and frequency simulations', Class. Quantum Grav. 21 991–998 (2003). 3

[Meers] Meers, B. J. and Strain, K. A.: 'Modulation, signal, and quantum noise in interferometers', Phys. Rev. A 44 4693–4703 (1991). 147, 169

[Mizuno] J. Mizuno: 'Comparison of optical configurations for laser-interferometric gravitational-wave-detectors', Ph.D. Thesis, University of Hannover (1995). 56

[Niebauer] Niebauer, T. M., Schilling, R., Danzmann, K., Rüdiger, A. and Winkler, W.: 'Nonstationary shot noise and its effect on the sensitivity of interferometers', Phys. Rev. A 43 5022–5029 (1991). 147, 169

[OSCAR] J. Degallaix: 'OSCAR: a Matlab based FFT code' (2008), available at http://www.mathworks.com/matlabcentral/fileexchange/20607-oscar 182, 185

[PyKat] D. Brown: 'PyKat Python interface and tools for FINESSE', http://www.gwoptics.org/pykat/. 151

[Rüdiger] A. Rüdiger: 'Phasenbeziehungen an einem symmetrischen Strahlteiler', internal note (1978). 32

[Siegman] A.E. Siegman: 'Lasers', University Science Books, Mill Valley (1986), see also the Errata at http://www-ee.stanford.edu/~siegman/lasers_book_errata.pdf 73, 75, 106, 193

[SimTools] Simtools, a collection of Matlab tools for optical simulations, available at http://www.gwoptics.org/simtools/ 94, 152

[Sparse] K.S. Kundert, A. Sangiovanni-Vincentelli: 'Sparse, A Sparse Linear Equation Solver', University of California, Berkeley (1988). 4

[VPB] J. Y. Vinet: 'The VIRGO physics book', this book is currently a very good optics textbook, available online at:
http://wwwcascina.virgo.infn.it/vpb 82, 179

[Willke01] B. Willke et al.: 'The GEO 600 gravitational wave detector', Class. Quantum Grav. 19 1377–1387 (2002). 3

[Yamamoto] H. Yamamoto: 'Mode matching and diffraction loss of FP cavity with thermal deformations', LIGO internal note, T0900306 (2010), https://dcc.ligo.org/LIGO-T0900306-v6 177