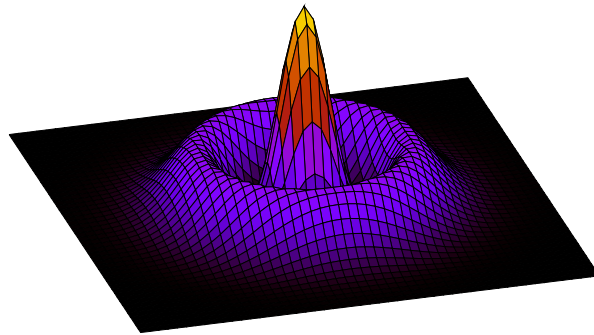


FINESSE 0.99.8

Frequency domain INterferomETER Simulation SoftwarE

Andreas Freise



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, shot-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.

21 Jan 2010

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
afreise@googlemail.com

Parts of the FINESSE source and 'mkat' have been written by Gerhard Heinzel, the document 'sidebands.ps' by Keita Kawabe, the Octave examples and its description by Gabriele Vajente, part of the FINESSE source have been written by Paul Cochrane.

THE SOFTWARE AND DOCUMENTATION IS PROVIDED AS IS WITHOUT ANY WARRANTY OF ANY KIND.

COPYRIGHT © BY ANDREAS FREISE 1999 – 2010.

For the moment I only distribute a binary version of the program. You may freely copy and distribute the program for non-commercial purposes only. Especially you should not charge fees or request donations for any part of the FINESSE distribution (or in connection with it) without the author's written permission. No other rights, such as ownership rights, are transferred.

```

** 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 batch filename : 'test.gnu'.

** 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
--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
-sparse, -klu : switch to SPARSE or KLU library respectively

** 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 ... )
isol name S node1 node2               - isolator
mod name f midx order am/pm [phase] node1 node2 - modulator
lens f node1 node2                   - thin lens

** 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
shot name node[*]                    - shot noise
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
qshot name num_demod f [phase] node[*] - quantum shotnoise detector
qshotS name num_demod f [phase] node[*] - quantum shotnoise sens.

** Available commands :
fsig name component [type] f phase [amp] - signal
tem input n m factor phase             - input power in TEMs
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 [angle] [mapname] filename - read mirror map file
savemap component mapname filename     - save coefficients to file
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
cav name component1 node component2 node - trace beam in cavity
startnode node                        - startnode of trace
retrace [off]                         - re-trace beam on/off

```

```

deriv_h value - set deriv_h for diff
phase 0-7 (default: 3) - change Gouy phases
(1: phi(00)=0, 2: gouy(00)=0, 4: switch ad phase)
** 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 x-axis for
3D plot
noxaxis - ignore xaxis commands
const name value - constant $name
variable name value - 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 - 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
pause - pauses after plotting
multi - plots all surfaces
GNU PLOT \ ... \ END - set of extra commands
for plotting.

```

```

FINESSE 0.99.8 - Help Screen (2) - A. Freise 21.01.2010

```

**** 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]

**** Geometrical conventions:**

tangential plane: x, z (index n), sagittal 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

**** 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

```

```

** 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)
** kmn n: 'n' bit coded word, set computation of coupling coeffs.:
    kmn 0 : use approximation from Bayer-Helms
    kmn 1 : verbose, i.e. print coupling coefficients.
    kmn 2 : use numerical intergration if x and y misalignment is set.
    kmn 4 : use numerical intergration if x or y misalignment is set.
    kmn 8 : use always numerical intergration.
              (default: kmn 0)
** bp, possible parameters for this detector:
    w : beam radius
    w0 : waist radius
    z : distance to waist
    zr : Rayleigh range
    g : Gouy phase
    q : Gaussian beam parameter
** isol S, suppression given in dB:
    amplitude coefficient computed as  $10^{-(S/20)}$ 
** maxtem 0/off : 0=n+m (TEM_nm) the order of TEM modes,
    'off' switches the TEM modus off explicitly

```


Contents

1	Introduction	1
1.1	Motivation	2
1.2	How does it work?	3
1.3	Quick start	3
1.3.1	Installation	4
1.3.2	How to perform a simulation	4
2	The program files	15
2.1	kat—the main program	15
2.2	kat.ini—the init file for kat	15
2.3	*.kat—the input files (how to do a calculation)	18
2.4	*.out—the output files	19
2.5	*.gnu—the Gnuplot batch files	19
2.6	*.m—the Matlab script files	20
3	Mathematical description of light beams and optical components	21
3.1	Introduction	21
3.1.1	Static response and frequency response	21
3.1.2	Transfer functions and error signals	22
3.1.3	The interferometer matrix	25
3.2	Conventions and concepts	26
3.2.1	Nodes and components	26
3.2.2	Mirrors and beam splitters	27
3.3	Frequencies and wavelengths	28
3.3.1	Phase change on reflection and transmission	29
3.3.2	Lengths and tunings	29
3.4	The plane-wave approximation	30
3.4.1	Description of light fields	31
3.4.2	Photodetectors and mixers	32
3.4.3	Modulation of light fields	32
3.4.4	Coupling of light field amplitudes	38
3.4.5	Input fields or the ‘right hand side’ vector	49
3.4.6	Photodetectors and demodulation	54
3.5	Shot-noise-limited sensitivity	57
3.5.1	Simple Michelson interferometer on a half fringe	58
3.5.2	Simple Michelson interferometer on a dark fringe	61
4	Higher-order spatial modes, the paraxial approximation	65
4.1	FINESSE with Hermite-Gaussian beams	65
4.2	Gaussian beams	66
4.3	Higher order Hermite-Gauss modes	68
4.3.1	Gaussian beam parameter	69

4.3.2	Tangential and sagittal plane	70
4.3.3	Gouy phase shift	71
4.3.4	ABCD matrices	71
4.4	Tracing the beam	74
4.5	Interferometer matrix with Hermite-Gauss modes	76
4.6	Coupling of Hermite-Gauss modes	77
4.6.1	Coupling coefficients for TEM modes	78
4.6.2	Mirror surface maps	81
4.6.3	Alignment transfer function	90
4.7	Detection of Hermite-Gauss modes	91
4.7.1	Amplitude detectors	91
4.7.2	Photodetectors	92
4.7.3	Beam detectors	92
4.8	Limits to the paraxial approximation	93
4.9	Mode mismatch in practice when using FINESSE	94
4.9.1	Phases and operating points	95
4.9.2	The <code>phase</code> command and its effects	95
4.9.3	Mode mismatch effects on the cavity phase	98
4.10	Misalignment angles at a beam splitter	100
4.11	Aperture effects and diffraction losses	102
5	Advanced Usage	105
5.1	FINESSE and Octave/Matlab	105
5.1.1	SimTools	106
5.1.2	Client-Server mode of FINESSE	106
5.1.3	Octave example scripts	113
5.2	Speed improvements	114
5.2.1	Higher-order modes	114
5.2.2	The <code>lock</code> command	115
5.2.3	KLU versus SPARSE	116
A	Example files	119
A.1	Simple examples	119
A.1.1	cavity1.kat	120
A.1.2	cavity2.kat	122
A.1.3	3D.kat	124
A.1.4	dc-schnupp.kat	126
A.1.5	donut.kat	128
A.1.6	rotate.kat	130
A.1.7	lock-cav1.kat	132
A.1.8	lock-cav2.kat	134
A.1.9	ring-cav.kat	136
A.2	mkat - a Perl preprocessor for FINESSE	138
B	Some mathematics	143
B.1	Hermite polynomials	143
B.2	The paraxial wave equation	143
C	Syntax reference	145
C.1	Comments	145
C.2	Components	146

C.3 Hermite-Gauss extension	151
C.4 Commands	159
C.5 Auxiliary plot commands	165
Acknowledgements	167
Bibliography	169

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.

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. 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.

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 [STAIC].

The idea for FINESSE was first raised in 1997, when I was visiting the MPQ in Garching, to assist Gerhard Heinzl 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 Heinzl 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 Heinzl 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 widely used in several projects; however, it has been most frequently utilised during the commissioning of GEO 600. Some of these simulation results have been published in [Freise03, Lück, Malec, Grote, Rakhmanov] and in [Freise].

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 file¹ 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 the Sparse package [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’.

Writing Gnuplot and Matlab batch files and starting Gnuplot [Gnuplot]: FINESSE uses Gnuplot to generate and display plots of the output data. 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 Matlab to plot the data. FINESSE saves a Matlab script file (extension ‘.m’) which can be called by Matlab to create a plot of the simulation results: 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.

1.3 Quick start

This section presents some example calculations with FINESSE. FINESSE it is very easy to use, despite this rather voluminous manual. The manual contains much basic information

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

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.

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 installed and FINESSE must know the correct command to start it. You can add the command for calling Gnuplot 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!

Recently Jan Harms has created a graphical user interface for FINESSE that can be used to generate the text input file by graphically adding, moving and connecting optical components [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 recommend LUXOR mostly for FINESSE beginners. In my opinion the careful use of text files is preferable when the interferometers or the simulation tasks become more complex. However, most of the manual does not refer to the user interface and is equally valid for text based or graphics based user interaction with FINESSE.

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

² The functionality of LUXOR does not necessarily include the latest features of the current version of FINESSE. Nevertheless, it should be well suited for the most simulation tasks.

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. See also Appendix A for more example files.

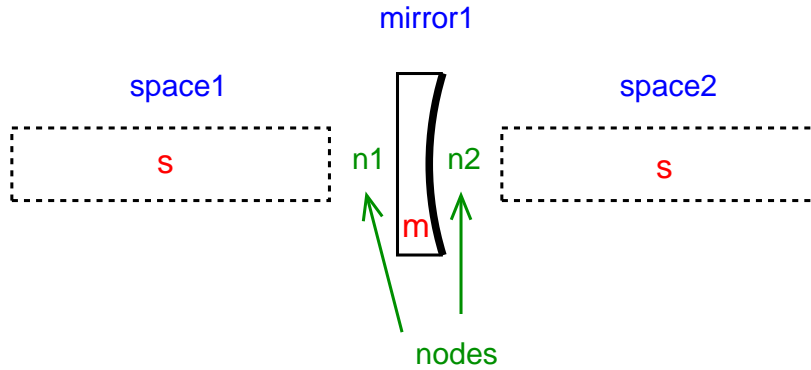


Figure 1.1: A mirror (m) and two spaces (s) connected via nodes $n1$ and $n2$.

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 1.1). 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 1.1, 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 (ϕ) (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.

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 output of Gnuplot.

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 C.**

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 a passing light beam in phase) 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 sometimes convenient for 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 be described by the following sum (see Section 3.4.3):

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

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

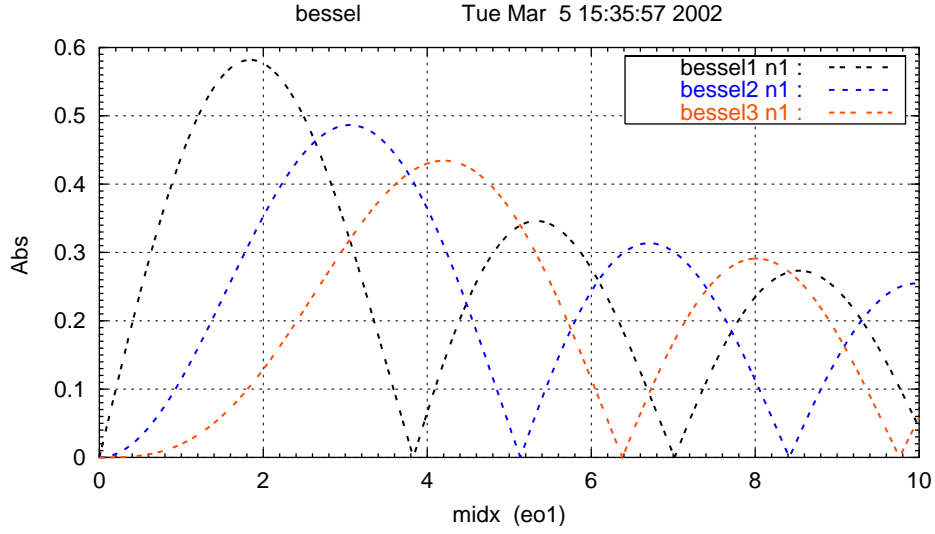


Figure 1.2: Simple example: testing the Bessel functions.

the above sum has to be taken into account:

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

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 bessell 40k n1
ad bessell 80k n1
ad bessell 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

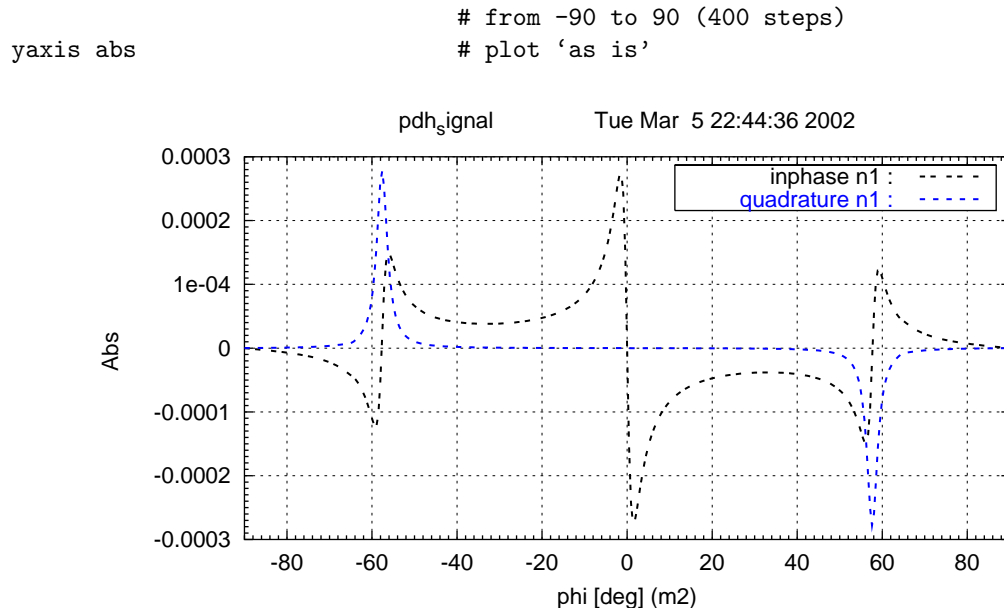


Figure 1.3: Second example: a Pound-Drever-Hall error signal.

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 1.3.

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 0.70
# (Transfer function 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 | |
#                               | |                               | |
#                               '-----'                       '-----'
#                               | |                               | |
#                               '._'                               '._'
#-----

m m1 0.9999 0.0001 0 n1 n2      # mirror R=0.9999 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
fsig sig1 m1 10 0          # signal inserted at m1
                             # f_sig=10Hz phase=0
pd2 inphase 40k 0 10 n1    # photo diode + 2 mixers:
                             # first demodulation 40kHz phase=0
                             # second at 10Hz (no phase ->
                             # network analyser mode)
xaxis sig1 f log .01 100 400 # tune f_sig from 0.01 to 100
put inphase f2 $x1          # put xaxis value also in f2 of the diode

yaxis db:deg                # plot gain in dB and phase
```

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
```

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 frequency³. 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

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

x_1 to the second frequency of the photodetector (f_2). The resulting plot is shown in Figure 1.4.

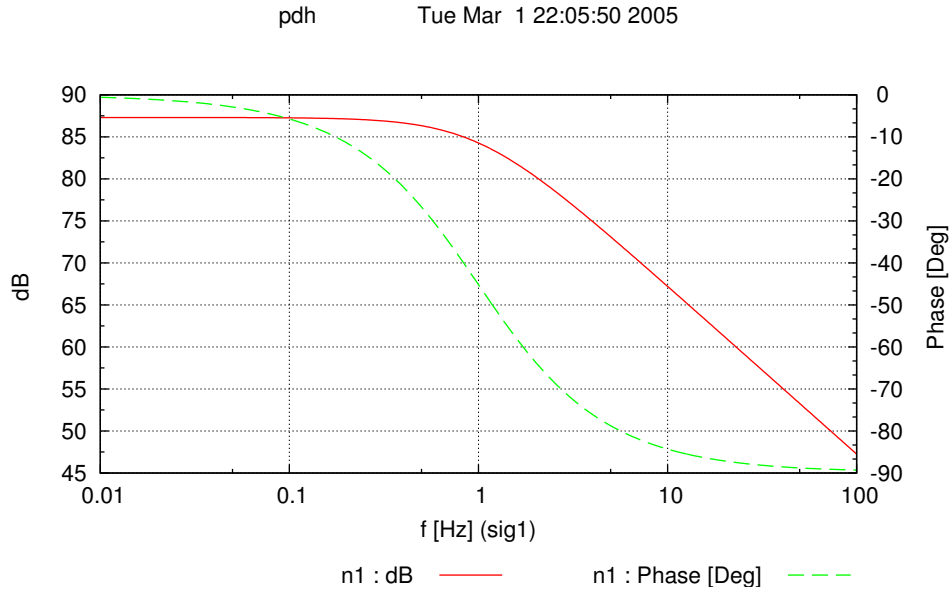


Figure 1.4: *Second example: the transfer function of the a Pound-Drever-Hall error signal (mirror motion to error signal).*

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 [5.1.2](#))
- perl1 : suppresses the printing of the banner and Gnuplot command
- 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 165)
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 Section C.3)
GNUCOMMAND : system command to start Gnuplot
GNUTERM : Gnuplot terminal description

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-31
qeff : 1.0
epsilon_c : 1.0
n0 : 1.0
gnuversion : 4.2
GNUCOMMAND : 'c:\programs\gnuplot\wgnuplot.exe' for Windows systems, 'gnu-
plot -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 addition, the file contains the photodetector definitions for horizontally split and vertically split detectors. Please see Appendix C.3 in the syntax reference for a description of how to add definitions of other detector types to the file.

In order to plot the data 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.

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 page 153 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
```

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.

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

1. '0 1 0 2 1.0', the beat between TEM₀₁ and TEM₀₂ is scaled by a factor of 1.0
2. '0 0 * 0 1.0', '*' means 'any': the beats of TEM₀₀ with TEM₀₀, TEM₁₀, TEM₂₀, TEM₃₀, 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 example files (see e.g. Appendix A). In addition, Appendix C 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 (see example file 'mach_zehnder.kat').

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.

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 only 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 the transfer function is calculated in a similar manner with the limitation that all *plants* are—of course—optical systems. The command:

```
fsg 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 modulation sidebands (i.e. those which are generated by a modulator). 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.

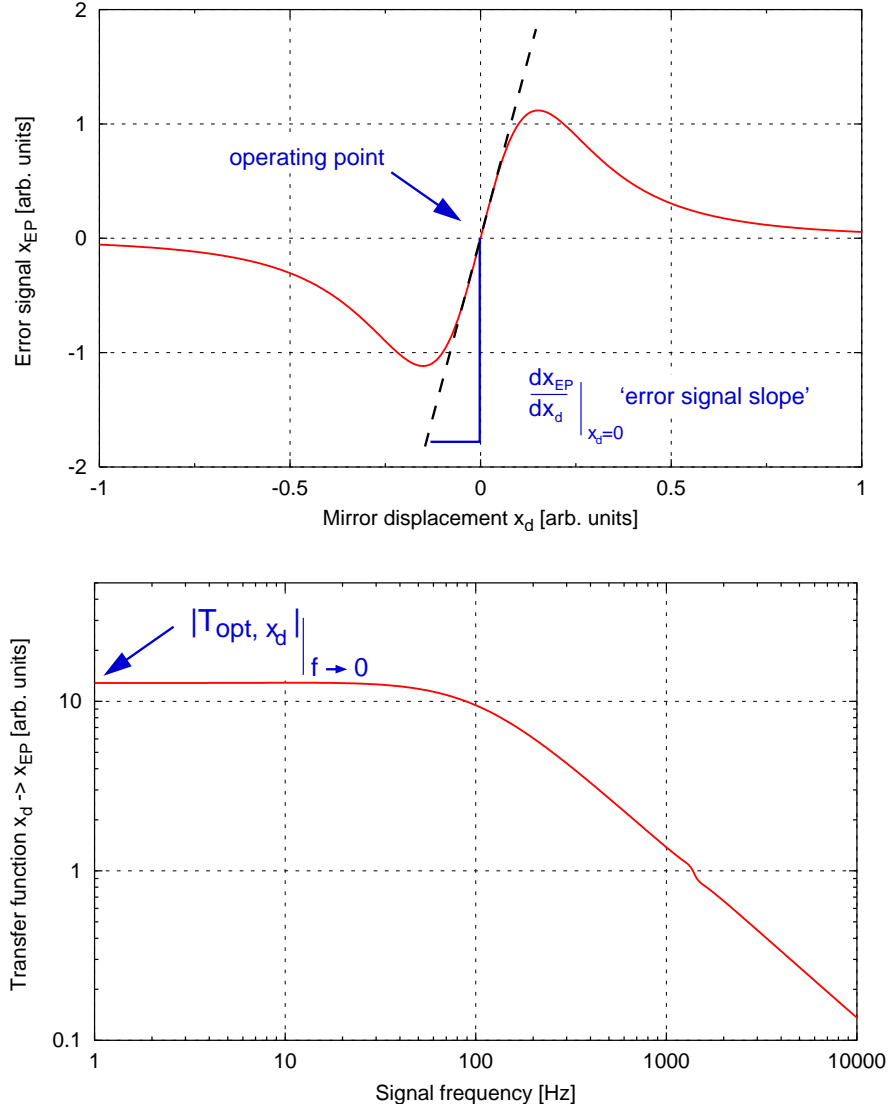


Figure 3.1: Example of an error signal: the top graph shows the electronic interferometer output signal as a function of mirror displacement. The operating point is given as the zero crossing, and the error-signal slope is defined as the slope at the operating point. The bottom graph shows the magnitude of the transfer function mirror displacement \rightarrow 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).

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:

$$x_d = 0 \quad \text{and} \quad x_{\text{EP}}(x_d = 0) = 0 \quad (3.1)$$

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

$$\tilde{x}_{\text{EP}}(f) = T_{\text{opt},x_d} T_{\text{det}} \tilde{x}_d(f), \quad (3.2)$$

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:

$$\left| \frac{dx_{\text{EP}}}{dx_d} \right|_{x_d=0} = |T_{\text{opt},x_d}|_{f \rightarrow 0} \quad (3.3)$$

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

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.

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

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.

Usually one interferometer matrix is sufficient to compute all light fields in one step. However, in order to decrease the size of the matrix and thus to increase the speed of the computation, FINESSE creates independent matrices for each frequency components and solves these sequentially. This has the disadvantage that couplings between different frequency components need to be described in a simplified form.

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’, ‘modulators’ and ‘signal frequencies’. The ‘modulators’ and ‘signal frequencies’ shift light power from a light field at a specified frequency to one or more field components with a frequency offset. Therefore, when independent matrices are constructed for each frequency, these components must 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, for each frequency that may be present in the interferometer, 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 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`

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

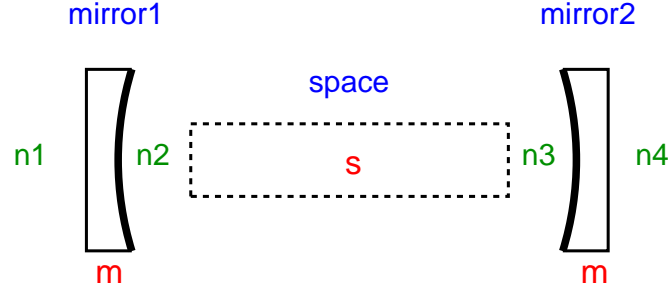


Figure 3.2: A simple Fabry-Perot cavity shown as components and nodes.

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 C).

3.2.2 Mirrors and beam splitters

The main components of interferometers in FINESSE are mirrors and beam splitters. Following the designs of the numerical methods used in FINESSE I have defined mirrors and beam splitters 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 it's use in reality but solely on 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 but **only** for signal modulation.

Please note that modulators cannot create sidebands of sidebands. This directly affects some simulation tasks. It is quite easy to make an error if this limitation is not considered. Two typical examples for analysis tasks that require some care are the transfer of frequency noise through an optical system or error signals for any kind of control scheme that needs double modulation (2 sets of RF sidebands, or higher harmonics of the RF sidebands). Please have a look at Keita Kawabe’s note on ‘Sidebands of Sidebands’ [Kawabe], which is part of the FINESSE package.

A modulator can be used as a phase or amplitude modulator. In both cases, modulators add symmetric sidebands to *input fields*, i.e. laser light and **not** to other modulation sidebands or signal sidebands. 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.

Please 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) of $f \approx 30 \cdot 10^6$ Hz,

and the signal sidebands at frequencies of 10 Hz to 1000 Hz¹.

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 $1\text{ }\mu\text{m}$ 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 wavelength². 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. Using plane waves, it is very simple to compute interferometer signals depending on the

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 $\lambda_0/2$.

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:

$$\vec{E}(t, \vec{x}) = \vec{E}_0 \cos(\omega t - \vec{k}\vec{x}). \quad (3.5)$$

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:

$$E(t) = E_0 \exp(i(\omega t + \varphi)) = a \exp(i\omega t), \quad (3.6)$$

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

$$\vec{E}(t) = \text{Re}\{E(t)\} \cdot \vec{e}_{\text{pol}}, \quad (3.7)$$

with \vec{e}_{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:

$$P = \epsilon_0 c E E^*, \quad (3.8)$$

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:

$$P = E E^*. \quad (3.9)$$

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

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:

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

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

$$E = E_0 \exp\left(i(\omega_0 t + m \cos(\omega_m t))\right). \quad (3.11)$$

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

$$E = E_0 \exp(i\omega_0 t) \sum_{k=-\infty}^{\infty} i^k J_k(m) \exp(ik\omega_m t). \quad (3.12)$$

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 $k\omega_m$. 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:

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

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

$$E = E_0 \exp(i\omega_0 t) \times \left(J_0(m) - iJ_{-1}(m) \exp(-i\omega_m t) + iJ_1(m) \exp(i\omega_m t) \right), \quad (3.14)$$

and with

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

we obtain:

$$E = E_0 \exp(i\omega_0 t) \left(1 + i \frac{m}{2} \left(\exp(-i\omega_m t) + \exp(i\omega_m t) \right) \right), \quad (3.16)$$

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:

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

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:

$$E = E_0 \exp\left(i\left(\omega_0 t + \frac{\Delta\omega}{\omega_m} \cos(\omega_m t)\right)\right), \quad (3.18)$$

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:

$$m = \frac{\Delta\omega}{\omega_m}. \quad (3.19)$$

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:

$$\begin{aligned} E &= E_0 \exp(i\omega_0 t) \left(1 + m \cos(\omega_m t)\right) \\ &= E_0 \exp(i\omega_0 t) \left(1 + \frac{m}{2} \exp(i\omega_m t) + \frac{m}{2} \exp(-i\omega_m t)\right). \end{aligned} \quad (3.20)$$

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:

$$\begin{aligned} E &= E_0 \exp(i\omega_0 t) \left(1 - \frac{m}{2} \left(1 - \cos(\omega_m t)\right)\right) \\ &= E_0 \exp(i\omega_0 t) \left(1 - \frac{m}{2} + \frac{m}{4} \exp(i\omega_m t) + \frac{m}{4} \exp(-i\omega_m t)\right). \end{aligned} \quad (3.21)$$

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:

$$A_{ssb} = A_0 - \frac{A_0 - A_m}{2}. \quad (3.22)$$

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:

$$E = E_0 \exp(i(\omega_0 t + m \cos(\omega_m t + \varphi_m(t))))). \quad (3.23)$$

Using Equation 3.17 phase noise can be expressed like this:

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

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

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

which results in the following field:

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

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:

$$E = E_0 e^{i\omega_0 t} \sum_{k=-order}^{order} i^k J_k(m) e^{ik\omega_m t} \times \left(1 + i \frac{k m_2}{2} e^{-i\omega_{noise} t} + i \frac{k m_2}{2} e^{i\omega_{noise} t} + O((km_2)^2) \right). \quad (3.27)$$

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

```
fsg signal1 eom1 10k 0
```

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

Oscillator amplitude noise³

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,

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

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

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

The light field now has form,

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

Using the identity,

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

³ This section has been contributed by Joshua Smith

we can obtain,

$$\begin{aligned}
 E &= E_0 \exp(i\omega_0 t) \\
 &\cdot \exp(im_1 \cos(\omega_{m_1} t)) \\
 &\cdot \exp\left(i \frac{m_1 m_2}{2} \cos((\omega_{m_1} + \omega_{m_2})t)\right) \\
 &\cdot \exp\left(i \frac{m_1 m_2}{2} \cos((\omega_{m_1} - \omega_{m_2})t)\right)
 \end{aligned} \tag{3.32}$$

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

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

with

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

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

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

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:

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

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:

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

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:

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

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

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

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.

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & s_1 \\ s_2 & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix} \quad \begin{array}{ccc} \xrightarrow{\text{In1}} & \text{---} & \xleftarrow{\text{In2}} \\ & \text{Space} & \\ \xleftarrow{\text{Out1}} & \text{---} & \xrightarrow{\text{Out2}} \end{array}$$

The propagation only affects the phase of the field:

$$s_1 = s_2 = \exp(-i\omega nL/c) = \exp(-i(\omega_0 + \Delta\omega)nL/c) = \exp(-i\Delta\omega nL/c), \quad (3.40)$$

where $\exp(-i\omega_0 nL/c) = 1$ 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:

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

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:

$$\phi = kx_m = x_m \frac{2\pi}{\lambda_0} = x_m \frac{\omega_0}{c}. \quad (3.42)$$

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:

$$\varphi = 2\phi \frac{\omega}{\omega_0}. \quad (3.43)$$

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:

$$\varphi_2 = -\varphi. \quad (3.44)$$

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:

$$E_{\text{trans}} = i t E_{\text{in}}, \quad (3.45)$$

with t as the amplitude transmittance of the mirror.

The coupling matrix for a mirror is:

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} m_{11} & m_{21} \\ m_{12} & m_{22} \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix} \quad \begin{array}{c} \xrightarrow{\text{In1}} \\ \xleftarrow{\text{Out1}} \end{array} \left[\begin{array}{c} \xleftarrow{\text{In2}} \\ \xrightarrow{\text{Out2}} \end{array} \right] \quad (3.46)$$

with the coefficients given as:

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

with $\phi = 2\pi \text{ phi}/360$, **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.

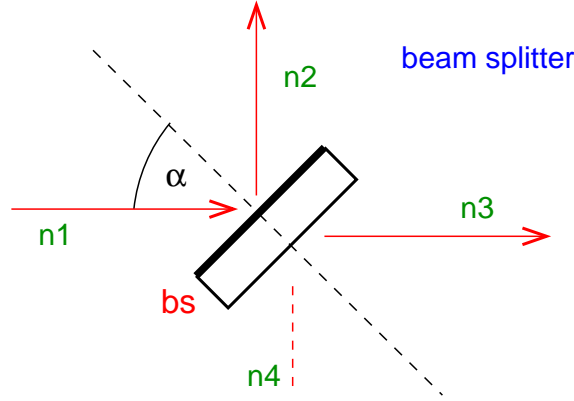


Figure 3.3: Beam splitter.

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 :

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

The coupling matrix has the following form:

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

with the coefficients:

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

and $\phi = 2\pi \text{ phi}/360$.

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:

$$eo_{12} = eo_{21} = 1. \quad (3.49)$$

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:

$$eo_{12} = eo_{21} = C, \quad (3.50)$$

with

$$C = 1 - \frac{m}{2}, \quad (3.51)$$

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

$$C = J_0(m), \quad (3.52)$$

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

$$C' = 1 - \frac{1 - C}{2}. \quad (3.53)$$

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:

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & d_{21} \\ d_{12} & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix} \quad \begin{array}{c} \xrightarrow{\text{In1}} \\ \xleftarrow{\text{Out1}} \end{array} \boxed{} \begin{array}{c} \xleftarrow{\text{In2}} \\ \xrightarrow{\text{Out2}} \end{array} \quad (3.54)$$

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.

$$\begin{pmatrix} \text{Out1} \\ \text{Out2} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \text{In1} \\ \text{In2} \end{pmatrix} \quad \begin{array}{c} \xrightarrow{\text{In1}} \\ \xleftarrow{\text{Out1}} \end{array} \left(\right) \begin{array}{c} \xleftarrow{\text{In2}} \\ \xrightarrow{\text{Out2}} \end{array}$$

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.

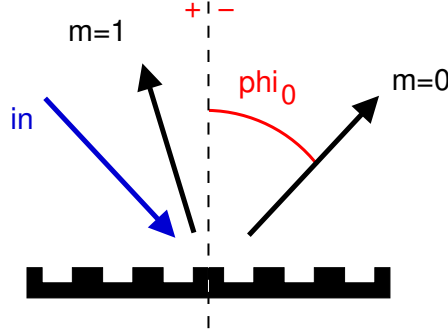


Figure 3.4: A grating illuminated by a beam (in). The number of outgoing beams is given by the grating equation Equation 3.55. The beams are numbered by an integer (m) and the angles with respect to the grating normal are given as ϕ_m (angles left of the grating normal are positive, right of the normal 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 ($m = -1, -2, \dots$) 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:

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

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:

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

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:

$$\frac{\lambda}{d} \leq \frac{2}{n}. \quad (3.57)$$

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

$$\frac{\lambda}{d} \leq \frac{1}{n}. \quad (3.58)$$

Table 3.1 gives an 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.

λ/d	m	0	1	2	3	4	-1	-2	-3	number of orders
$]2, \text{inf}]$		x								1
$]1, 2]$		x	x							2
$]2/3, 1]$		x	x	x			x			4
$]1/2, 2/3]$		x	x	x	x		x			5
$]1/3, 1/2]$		x	x	x	x	x	x	x		7
$]3/5, 1/3]$		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 n th order Littrow configuration is given by:

$$\phi_n = \alpha, \quad (3.59)$$

which yields:

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

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:

$$b_i = A_{ij}a_j, \quad (3.61)$$

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

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

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:

$$\lambda/2 < d < 3/2 \lambda. \quad (3.63)$$

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

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

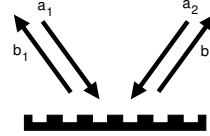
The angle of incidence is set automatically by FINESSE (the positive value is chosen by default).

The two coupling efficiencies η_0, η_1 are constrained by energy conservation⁴ (as for the beam splitter) as:

$$\eta_0^2 + \eta_1^2 = 1. \quad (3.65)$$

The coupling matrix is given by:

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} i\eta_1 & \eta_0 \\ \eta_0 & i\eta_1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$



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

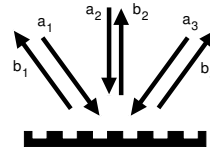
$$\lambda < d < 2\lambda. \quad (3.66)$$

And ϕ_2 must be equal to α . This yields:

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

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

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} \eta_2 e^{i\phi_2} & \eta_1 e^{i\phi_1} & \eta_0 e^{i\phi_0} \\ \eta_1 e^{i\phi_1} & \rho_0 e^{i\phi_0} & \eta_1 e^{i\phi_1} \\ \eta_0 e^{i\phi_0} & \eta_1 e^{i\phi_1} & \eta_2 e^{i\phi_2} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$$



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 by inserting extra mirrors with $R = 0, T \neq 1$.

with:

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

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

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

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:

$$a + b \neq 2c. \quad (3.78)$$

As before we can write this as

$$a < 2c - 1 \quad \vee \quad a > 2c + 1. \quad (3.79)$$

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 possible values for c even further. Combining Equation 3.77 and Equation 3.79 we get:

$$a > 1 - c \quad \wedge \quad a < 2c - 1. \quad (3.80)$$

This is only possible if

$$1 - c < 2c - 1, \quad (3.81)$$

and thus $c > 2/3$.

In summary we get:

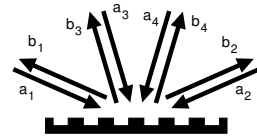
$$\begin{aligned} \lambda/2 &< d < 3/2 \lambda, \\ 1 - \lambda/d &< \sin(\alpha) < 2\lambda/d - 1 \quad \text{for} \quad \lambda/d < 1, \\ \lambda/d - 1 &< \sin(\alpha) \quad \text{for} \quad \lambda/d > 1. \end{aligned} \quad (3.82)$$

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:

$$\eta_0^2 + \eta_1^2 = 1. \quad (3.83)$$

The coupling matrix is given by:

$$\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 0 & A_{21} & A_{31} & 0 \\ A_{12} & 0 & 0 & A_{42} \\ A_{13} & 0 & 0 & A_{43} \\ 0 & A_{24} & A_{34} & 0 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$



with the coefficients:

$$A_{12} = A_{21} = \eta_0, \quad (3.84)$$

$$A_{13} = A_{31} = i\eta_1, \quad (3.85)$$

$$A_{24} = A_{42} = i\eta_1, \quad (3.86)$$

$$A_{34} = A_{43} = \eta_0. \quad (3.87)$$

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:

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

with

$$P = \epsilon_0 c |a|^2, \quad (3.89)$$

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:

$$a_{in} = \frac{m}{4}, \quad (3.90)$$

and for phase modulation:

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

with φ given as (Equation 3.12):

$$\varphi = \pm k \cdot \left(\frac{\pi}{2} + \varphi_s\right), \quad (3.92)$$

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 ($f_{\text{mod}} < 0$) we get $\varphi = -\dots$, for ‘upper’ sidebands ($f_{\text{mod}} > 0$) it is $\varphi = +\dots$.

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, artificial 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 amplitude of 1. The numerical value of 1 has a different meaning for applying signals to different components (see below). The amplitude that can be 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:

$$E_{in} = E_0 \exp(i\omega_c t + \varphi_c), \quad (3.93)$$

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:

$$E_{out} = A E_0 \exp(i\omega_c t + \varphi_c + \varphi(t) + B), \quad (3.94)$$

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

$$\varphi(t) = m \cos(\omega_s t + \varphi_s), \quad (3.95)$$

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:

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

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:

$$\begin{aligned} E_{sb} &= i \frac{m}{2} A E_0 \exp(i((\omega_c \pm \omega_s)t + \varphi_c + B \pm \varphi_s)) \\ &= \frac{m}{2} A E_0 \exp(i(\omega_{sb}t + \pi/2 + \varphi_c + B \pm \varphi_s)). \end{aligned} \quad (3.97)$$

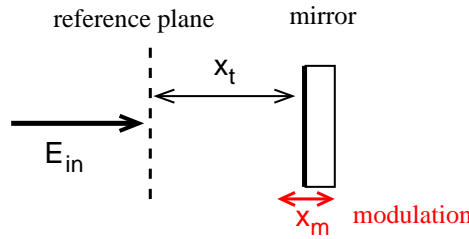


Figure 3.5: Signal applied to a mirror: modulation of the mirror position.

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:

$$E_{\text{in}} = E_0 \exp(i(\omega_c t + \varphi_c - k_c x)) = E_0 \exp(i\omega_c t + \varphi_c). \quad (3.98)$$

If the mirror is detuned by x_t (here given in meters) then the electric field at the mirror is:

$$E_{\text{mir}} = E_{\text{in}} \exp(-i k_c x_t). \quad (3.99)$$

With the given parameters for the signal frequency, the position modulation can be written as $x_m = a_s \cos(\omega_s t + \varphi_s)$ and thus the reflected field at the mirror is:

$$E_{\text{refl}} = r E_{\text{mir}} \exp(i 2 k_c x_m) = r E_{\text{mir}} \exp(i 2 k_c a_s \cos(\omega_s t + \varphi_s)), \quad (3.100)$$

setting $m = 2 k_c a_s$, this can be expressed as:

$$\begin{aligned} E_{\text{refl}} &= r E_{\text{mir}} \left(1 + i \frac{m}{2} \exp(-i(\omega_s t + \varphi_s)) + i \frac{m}{2} \exp(i(\omega_s t + \varphi_s)) \right) \\ &= r E_{\text{mir}} \left(1 + \frac{m}{2} \exp(-i(\omega_s t + \varphi_s - \pi/2)) \right. \\ &\quad \left. + \frac{m}{2} \exp(i(\omega_s t + \varphi_s + \pi/2)) \right). \end{aligned} \quad (3.101)$$

This gives an amplitude for both sidebands of:

$$a_{\text{sb}} = r m/2 E_0 = r k_c a_s E_0. \quad (3.102)$$

The phase back at the reference plane is:

$$\varphi_{\text{sb}} = \varphi_c + \frac{\pi}{2} \pm \varphi_s - (k_c + k_{\text{sb}}) x_t, \quad (3.103)$$

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:

$$\begin{aligned} \varphi_{\text{sb}} &= \varphi_c + \frac{\pi}{2} \pm \varphi_s - (\omega_c + \omega_{\text{sb}})/c x_t \\ &= \varphi_c + \frac{\pi}{2} \pm \varphi_s - (\omega_c + \omega_{\text{sb}})/c \lambda_0/360 \phi \\ &= \varphi_c + \frac{\pi}{2} \pm \varphi_s - (\omega_c + \omega_{\text{sb}})/\omega_0 2\pi/360 \phi. \end{aligned} \quad (3.104)$$

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

$$a_{\text{sb}} = r k_c E_0 = r \omega_c/c E_0 = r 2\pi/\lambda_0 E_0 \approx 6 \cdot 10^6. \quad (3.105)$$

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:

$$\begin{aligned} E_{\text{refl}} &= r E_{\text{mir}} \exp(i 2\omega_c/\omega_0 x_m) \\ &= r E_{\text{mir}} \exp\left(i 2\omega_c/\omega_0 a'_s \cos(\omega_s t + \varphi_s)\right), \end{aligned} \quad (3.106)$$

and thus the sideband amplitudes are:

$$a_{\text{sb}} = r \omega_c/\omega_0 a'_s E_0, \quad (3.107)$$

with the factor ω_c/ω_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 $x_m \rightarrow -x_m$ and $x_t \rightarrow -x_t$, yielding the same amplitude as for the reflection at the front surface, but with a slightly different phase:

$$\begin{aligned} \varphi_{\text{sb,back}} &= \varphi_c + \frac{\pi}{2} \pm (\varphi_s + \pi) + (k_c + k_{\text{sb}}) x_t \\ &= \varphi_c + \frac{\pi}{2} \pm (\varphi_s + \pi) + (\omega_c + \omega_{\text{sb}})/\omega_0 2\pi/360 \phi. \end{aligned} \quad (3.108)$$

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:

$$a_{sb} = r \frac{\omega_c}{\omega_0} a_s \cos(\alpha) E_0, \quad (3.109)$$

$$\phi_{sb,front} = \varphi_c + \frac{\pi}{2} \pm \varphi_s - (k_c + k_{sb})x_t \cos(\alpha), \quad (3.110)$$

$$\phi_{sb,back} = \varphi_c + \frac{\pi}{2} \pm (\varphi_s + \pi) + (k_c + k_{sb})x_t \cos(\alpha). \quad (3.111)$$

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]):

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

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:

$$a_{sb} = \frac{1}{4} \frac{\omega_c}{\omega_s} \sin \left(\omega_s \frac{n L}{c} \right) a_s E_0, \quad (3.113)$$

$$\phi_{sb} = \varphi_c + \frac{\pi}{2} \pm \varphi_s - (\omega_c + \omega_s) \frac{nL}{c}. \quad (3.114)$$

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

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

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

$$\begin{aligned} a_{sb} &= \frac{a_s}{2\omega_s} E_0, \\ \phi_{sb} &= \varphi_c + \frac{\pi}{2} \pm \varphi_s. \end{aligned} \quad (3.116)$$

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:

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

with

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

The sideband amplitudes are:

$$\begin{aligned} a_{sb} &= \frac{a_s k m_2}{2} E_{\text{mod}}, \\ \phi_{sb} &= \varphi_{\text{mod}} + \frac{\pi}{2} \pm \varphi_s. \end{aligned} \quad (3.119)$$

3.4.6 Photodetectors and demodulation

With all the different ways of plotting signals in FINESSE it is important to understand that every photodiode output represents only **one** frequency component of a signal. The ‘signal’ can be a light field, a light power, or a mixer output. Of course, you can calculate more than one frequency component at a time, but only by specifying different outputs 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:

$$C_{\text{ampere}} = \frac{e q_{\text{eff}} \lambda_0}{hc} \left[\frac{\text{A}}{\text{W}} \right], \quad (3.120)$$

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 $\text{m}/\sqrt{\text{Hz}}$.

- `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

$$E = e^{i\omega_0 t} \sum_{n=0}^N a_n e^{i\omega_n t}, \quad (3.121)$$

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

$$E = e^{i\omega_0 t} (b_0 + b_1 e^{i\omega_1 t} + b_{-1} e^{-i\omega_1 t} + \dots + b_M e^{i\omega_M t} + b_{-M} e^{-i\omega_M t}), \quad (3.122)$$

where ω_0 is the carrier frequency and $\omega_1, \omega_2, \dots, \omega_M > 0$ 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:

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

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:

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

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:

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

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

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

multiplied with the local oscillator it becomes

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

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

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

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

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

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:

$$\begin{aligned} S_{1,\text{DC,phase}} &= \sum_{ij} \text{Re} \{A_{ij}\}, \\ S_{1,\text{DC,quad}} &= \sum_{ij} \text{Im} \{A_{ij}\}. \end{aligned} \quad (3.130)$$

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

$$x = S_{1,\text{DC}}. \quad (3.131)$$

If no phase is given the output is a complex number

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

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:

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

This can be written as

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

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:

$$\begin{aligned} z_1 &= \sum_{ij} A_{ij} \quad \text{with} \quad \{i, j \mid i, j \in \{0, \dots, N\} \wedge \omega_i - \omega_j = \omega_+\}, \\ z_2 &= \sum_{ij} A_{kl} \quad \text{with} \quad \{k, l \mid k, l \in \{0, \dots, N\} \wedge \omega_k - \omega_l = \omega_-\}. \end{aligned} \quad (3.135)$$

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

$$x = \text{Re} \{ (z_1 e^{-i\varphi_x} + z_2 e^{i\varphi_x}) e^{-i\varphi_y} \}. \quad (3.136)$$

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:

$$z = z_1 e^{-i\varphi_x} + z_2 e^{i\varphi_x}. \quad (3.137)$$

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 Shot-noise-limited sensitivity

FINESSE allows one to compute the shot-noise-limited sensitivity using the command `pdS`. However, to date this shot noise computation must be considered as a simple approximation; it only gives correct results in the absence of modulation sidebands and shot-noise dominated interferometers, i.e. where radiation pressure effects can be neglected.

FINESSE is currently being extended to correctly compute the quantum noise of the light upon detection with a photodiode.

This section gives a short description on how FINESSE computes shot-noise and the linear spectral density that is often called sensitivity.

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} :

$$S_I(f) = 2 e \bar{I}, \quad (3.138)$$

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:

$$\bar{I} = eN = \frac{e \eta \lambda}{\hbar 2\pi c} \bar{P}, \quad (3.139)$$

with N as the number of photons and η the quantum efficiency of the diode. Instead 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 $S_P(f) = C_2^2 S_I(f)$. With the relation $S_I(f) = C_1 \bar{I}$ we can then write:

$$S_P(f) = C_2^2 S_I(f) = C_2^2 C_1 \bar{I} = C_2^2 C_1 \bar{P} / C_2 = 2 \frac{2\pi \hbar c}{\lambda} \bar{P}. \quad (3.140)$$

3.5.1 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 $P_0 = 1$ W,
- the quantum efficiency is $\text{qeff} = \eta = 1$,
- 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 $L_{\text{north}} = 1201$ m and $L_{\text{east}} = 1200$ 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 3.140 we expect a value of:

$$S_P(f) = \frac{6.6262 \cdot 10^{-34} \cdot 299792458}{1064 \cdot 10^{-9}} \text{ W}^2/\text{Hz} = 1.867 \cdot 10^{-19} \text{ W}^2/\text{Hz}, \quad (3.141)$$

or as a linear spectral density $\sqrt{S_P} = 4.321 \cdot 10^{-10} \text{ W}/\sqrt{\text{Hz}}$. 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:

$$E_{N \text{ in}} = \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{4}\right), \quad (3.142)$$

$$E_{E \text{ in}} = \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{2}\right). \quad (3.143)$$

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:

$$\varphi_{\text{sb}} = \varphi_c + \frac{\pi}{2} \pm \varphi_s - (k_c + k_{\text{sb}}) x_t. \quad (3.144)$$

The light reflected by the end mirrors (with $r = 1$) can then be written as:

$$\begin{aligned} E_N &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{4}\right) \left(1 + a_s \exp\left(i \omega_s t + \frac{\pi}{2}\right) + a_s \exp\left(-i \omega_s t - \frac{\pi}{2}\right)\right) \\ &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{4}\right) \left(1 + a_s i \left(\exp\left(i \omega_s t\right) + \exp\left(-i \omega_s t\right)\right)\right) \\ &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{4}\right) \left(1 + 2a_s i \cos(\omega_s t)\right), \end{aligned} \quad (3.145)$$

$$\begin{aligned} E_E &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{2}\right) \left(1 + a_s \exp\left(i \omega_s t + \frac{3\pi}{2}\right) + a_s \exp\left(-i \omega_s t + \frac{\pi}{2}\right)\right) \\ &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp\left(i \frac{\pi}{2}\right) \left(1 - 2a_s i \cos(\omega_s t)\right), \end{aligned}$$

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:

$$\begin{aligned}
 E_S &= \frac{1}{\sqrt{2}} \exp(i \frac{\pi}{2}) E_N + \frac{1}{\sqrt{2}} \exp(-i \frac{\pi}{4}) E_E \\
 &= \frac{1}{2} E_{\text{in}} \exp(i \frac{\pi}{4}) (1 + i - 2a_s i \cos(\omega_s t) - 2a_s \cos(\omega_s t)) \\
 &= \frac{1}{2} E_{\text{in}} \exp(i \frac{\pi}{4}) (1 + i) (1 - 2a_s \cos(\omega_s t)) \\
 &= \frac{i}{\sqrt{2}} E_{\text{in}} (1 - 2a_s \cos(\omega_s t)).
 \end{aligned} \tag{3.146}$$

With $|E_{\text{in}}|^2 = P_0 = 1 \text{ W}$ the power detected by the diode in each output port is thus:

$$\begin{aligned}
 I_{\text{out}} &= \frac{1}{2} P_0 (1 + 4a_s^2 \cos^2(\omega_s t) - 4a_s \cos(\omega_s t)) \\
 &= \frac{1}{2} (1 + 4a_s^2 \cos^2(\omega_s t) - 4a_s \cos(\omega_s t)) \text{ [W]}.
 \end{aligned} \tag{3.147}$$

The power in the signal sidebands can be neglected so that the DC power in both output ports is given as $P_0/2 = 0.5 \text{ W}$.

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:

$$\begin{aligned}
 I_{\text{demod}} &= 2a_s P_0 \cos^2(\omega_s t) + O(\omega) + O(3\omega) \\
 &= a_s P_0 + a_s P_0 \cos(2\omega_s t) + O(\omega) + O(3\omega).
 \end{aligned} \tag{3.148}$$

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:

$$T_{\Delta L \rightarrow P} = \frac{2\pi P_0}{\lambda} = \frac{2\pi}{1064 \cdot 10^{-9}} \text{ W/m.} \quad (3.149)$$

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:

$$T_{\Delta L' \rightarrow P} = \frac{\pi P_0}{\lambda} = \frac{\pi}{1064 \cdot 10^{-9}} \text{ W/m.} \quad (3.150)$$

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 3.140 by the transfer function:

$$\sqrt{S_{\Delta L'}(f)} = \frac{\sqrt{\frac{2\pi \hbar c}{\lambda} P_0}}{\frac{\pi P_0}{\lambda}} = \sqrt{2 \frac{\hbar c \lambda}{\pi P_0}} = 1.463 \cdot 10^{-16} \text{ m}/\sqrt{\text{Hz}}. \quad (3.151)$$

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 $\sqrt{S_{\Delta L}(f)} = \sqrt{\frac{\hbar c \lambda}{\pi P_0}} = 1.034 \cdot 10^{-16} \text{ m}/\sqrt{\text{Hz}}$.

3.5.2 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-limited 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:

$$\begin{aligned} E_{N\text{in}} &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp(i \frac{\pi}{2}), \quad \text{and} \\ E_{E\text{in}} &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp(i \frac{\pi}{2}). \end{aligned} \quad (3.152)$$

The light reflected by the end mirrors (with $r = 1$) can then be written as:

$$\begin{aligned} E_N &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp(i \frac{\pi}{2}) (1 + a_s i (\exp(i \omega_s t) + a_s \exp(-i \omega_s t))) \\ &= \frac{1}{\sqrt{2}} E_{\text{in}} (i - 2a_s \cos(\omega_s t)), \\ E_E &= \frac{1}{\sqrt{2}} E_{\text{in}} \exp(i \frac{\pi}{2}) (1 - a_s i (\exp(i \omega_s t) + a_s \exp(-i \omega_s t))) \\ &= \frac{1}{\sqrt{2}} E_{\text{in}} (i + 2a_s \cos(\omega_s t)). \end{aligned} \quad (3.153)$$

And hence in the south port we get:

$$\begin{aligned} E_S &= \frac{1}{\sqrt{2}} \exp(i \frac{\pi}{2}) E_N + \frac{1}{\sqrt{2}} \exp(-i \frac{\pi}{2}) E_E \\ &= -2i E_{\text{in}} a_s \cos(\omega_s t). \end{aligned} \quad (3.154)$$

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:

$$E_S = i b (\exp(i \omega_m t) + \exp(-i \omega_m t)) - 2i E_{\text{in}} a_s \cos(\omega_s t) \quad (3.155)$$

$$= 2i b \cos(\omega_m t) - 2i E_{\text{in}} a_s \cos(\omega_s t). \quad (3.156)$$

The power detected by the diode in each output port is thus:

$$I_{\text{out}} = 4b^2 \cos^2(\omega_m t) + 4a_s^2 E_{\text{in}}^2 \cos^2(\omega_s t) - 8ba_s E_{\text{in}} \cos(\omega_m t) \cos(\omega_s t) \quad (3.157)$$

$$= 2b^2 + 2a_s^2 P_0 - 8ba_s \sqrt{P_0} \cos(\omega_m t) \cos(\omega_s t) + O(2\omega_m) + O(2\omega_s). \quad (3.158)$$

$$(3.159)$$

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

$$\begin{aligned}\sqrt{S_P(f)} &= \sqrt{2 \frac{2\pi\hbar c}{\lambda} 2b^2 / \sqrt{\text{Hz}}} \\ &= 8.962 \cdot 10^{-12} \text{ W} / \sqrt{\text{Hz}}.\end{aligned}\tag{3.160}$$

The shot detector in FINESSE gives $8.978\text{e-}12$. 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:

$$T_{\Delta L' \rightarrow P} = \frac{2\pi}{\lambda} b \sqrt{P_0}.\tag{3.161}$$

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:

$$\left[\sqrt{S_P(f)} \cos(\omega_m t) \right]_{|_{DC}} \approx (A_1 \cos(-\omega_m t) + A_2 \cos(\omega_m t)) \cos(\omega_m t)\tag{3.162}$$

with A_1 and A_2 as two uncorrelated amplitudes⁵ of the magnitude $\sqrt{S_P(f)}$. The right side of above equation yields:

$$\frac{1}{2}(A_1 + A_2) = \frac{1}{\sqrt{2}}A_1 = \frac{1}{\sqrt{2}}\sqrt{S_P(f)} = \sqrt{\frac{2\pi\hbar c}{\lambda} 2b^2 / \sqrt{\text{Hz}}}\tag{3.163}$$

⁵ 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:

$$\begin{aligned}\sqrt{S_{\Delta L}(f)} &= \frac{\sqrt{\frac{2\pi\hbar c}{\lambda} 2b^2}}{\frac{2\pi}{\lambda} b\sqrt{P_0}} = \sqrt{\frac{\hbar c\lambda}{\pi P_0}} \\ &= 1.0341010^{-16} \text{ m}/\sqrt{\text{Hz}},\end{aligned}\tag{3.164}$$

which is the same as in the case of the Michelson interferometer at a half fringe with two detectors.

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 $n + m \leq \text{order}$.

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 [C.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:

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

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 ($t = 0$):

$$E(x, y, z) = \exp(-ikz) \sum_{n,m} a_{nm} u_{nm}(x, y, z). \quad (4.2)$$

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 [B.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 [4.8](#) 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:

$$\begin{aligned} E(x, y, z) &= E_0 u_{00} \exp(-ikz) \\ &= E_0 \left(\frac{1}{R_C(z)} - i \frac{\lambda}{\pi w^2(z)} \right) \cdot \exp \left(-ik \frac{x^2 + y^2}{2R_C(z)} - \frac{x^2 + y^2}{w^2(z)} - ikz \right). \end{aligned} \quad (4.3)$$

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:

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

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.

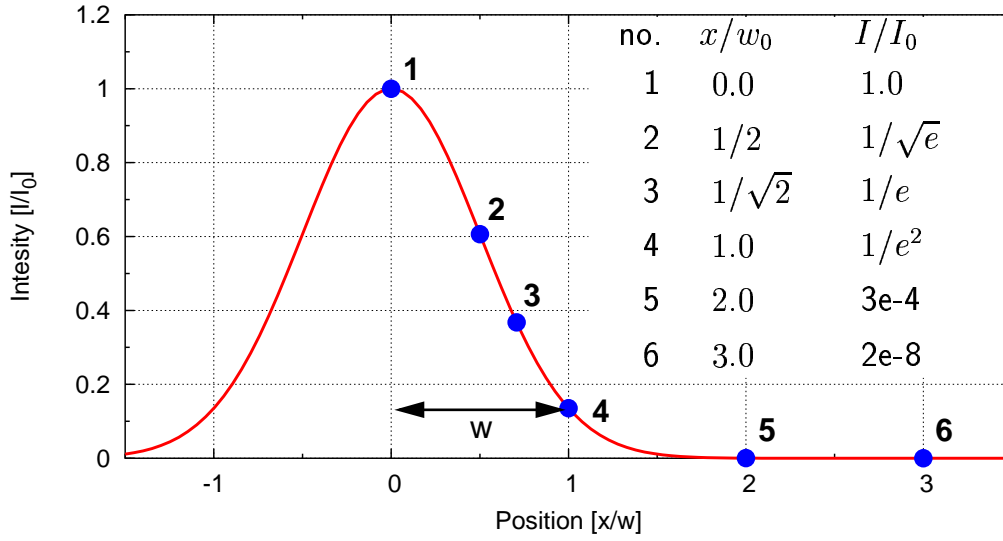


Figure 4.1: One dimensional cross-section of a Gaussian beam. The width of the beam is given by the radius w at which the intensity is $1/e^2$ of the maximum intensity.

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:

$$z_R = \frac{\pi w_0^2}{\lambda}. \quad (4.5)$$

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

$$w(z) = w_0 \sqrt{1 + \left(\frac{z - z_0}{z_R} \right)^2}. \quad (4.6)$$

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

$$w(z) \approx w_0 \frac{z}{z_R} = \frac{z\lambda}{\pi w_0}. \quad (4.7)$$

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

$$\Theta = \arctan\left(\frac{w_0}{z_R}\right) = \arctan\left(\frac{\lambda}{\pi w_0}\right) \approx \frac{w_0}{z_R}. \quad (4.8)$$

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 function of z :

$$R_C(z) = z - z_0 + \frac{z_R^2}{z - z_0}. \quad (4.9)$$

For the radius of curvature we also find:

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

4.3 Higher order Hermite-Gauss modes

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

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

with n, m being the *mode numbers* or *mode indices*. In this case n refers to the modes in the y - z plane (sagittal) 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 B.1,
- $w(z)$: beam radius or spot size,
- $R_C(z)$: 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:

$$\iint dx dy u_{nm} u_{n'm'}^* = \delta_{nn'} \delta_{mm'}. \quad (4.12)$$

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

$$P = \sum_{n,m} a_{nm} a_{nm}^*. \quad (4.13)$$

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

$$P = \sum_{n,m} \sum_i \sum_j a_{inm} a_{jnm}^* \quad \text{with} \quad \{i, j \mid i, j \in \{0, \dots, N\} \wedge \omega_i = \omega_j\}. \quad (4.14)$$

The x and y dependencies can be separated so that:

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

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:

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

and can also be written as:

$$q(z) = i z_R + z - z_0 = q_0 + z - z_0, \quad \text{where} \quad q_0 = i z_R. \quad (4.17)$$

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:

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

The complete set of solutions as given in Equation 4.11 can now be written as²:

$$u_{nm}(x, y, z) = u_n(x, z) u_m(y, z), \quad (4.19)$$

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

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

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 :

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

and

$$R_C(z) = \frac{|q|^2}{\text{Re}\{q\}}. \quad (4.22)$$

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-negligible 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 4.8 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:

$$u_{nm}(x, y) = u_n(x, q_t) u_m(y, q_s). \quad (4.23)$$

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:

$$\Psi(z) = \arctan\left(\frac{z - z_0}{z_R}\right), \quad (4.24)$$

or, using the Gaussian beam parameter:

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

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

$$\varphi = (n + m + 1)\Psi(z). \quad (4.26)$$

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

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

with

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

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 C.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:

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

with the coefficient matrix,

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

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:

$$M = \begin{pmatrix} 1 & 0 \\ \frac{n_2 - n_1}{R_C} & 1 \end{pmatrix} \quad \begin{array}{c} q_1 \rightarrow \\ n_1 \end{array} \left| \begin{array}{c} \text{mirror} \\ n_2 \end{array} \right. \begin{array}{c} q_2 \rightarrow \end{array} \quad (4.31)$$

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:

$$M = \begin{pmatrix} 1 & 0 \\ -\frac{2n_1}{R_C} & 1 \end{pmatrix} \quad \begin{array}{c} q_1 \rightarrow \\ q_2 \leftarrow \\ n_1 \end{array} \left| \begin{array}{c} \text{mirror} \\ n_2 \end{array} \right. \quad (4.32)$$

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):

$$M_t = \begin{pmatrix} \frac{\cos(\alpha_2)}{\cos(\alpha_1)} & 0 \\ \frac{\Delta n}{R_C} & \frac{\cos(\alpha_1)}{\cos(\alpha_2)} \end{pmatrix} \quad \begin{array}{c} q_1 \rightarrow \\ q_2 \rightarrow \end{array} \quad (4.33)$$

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

with α_2 given by Snell's law:

$$n_1 \sin(\alpha_1) = n_2 \sin(\alpha_2), \quad (4.34)$$

and Δn by:

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

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:

$$\begin{aligned} A &\longrightarrow 1/A, \\ D &\longrightarrow 1/D. \end{aligned} \quad (4.36)$$

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

$$\begin{aligned} M_t &= \begin{pmatrix} 1 & 0 \\ -\frac{2n_1}{R_C \cos(\alpha_1)} & 1 \end{pmatrix} \\ M_s &= \begin{pmatrix} 1 & 0 \\ -\frac{2n_1 \cos(\alpha_1)}{R_C} & 1 \end{pmatrix} \end{aligned} \quad \begin{array}{c} \text{Diagram: A beam splitter interface between medium } n_1 \text{ (left) and } n_2 \text{ (right). An incident ray from } n_1 \text{ at angle } \alpha_1 \text{ splits into a reflected ray } q_2 \text{ and a transmitted ray } q_1. \end{array} \quad (4.37)$$

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

$$\begin{aligned} R_C &\longrightarrow -R_C, \\ n_1 &\longrightarrow n_2, \\ \alpha_1 &\longrightarrow -\alpha_2. \end{aligned} \quad (4.38)$$

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

$$M = \begin{pmatrix} 1 & 0 \\ -\frac{1}{f} & 1 \end{pmatrix} \quad \begin{array}{c} \text{Diagram: A thin lens represented by a vertical oval. An incident ray } q_1 \text{ enters from the left and an outgoing ray } q_2 \text{ exits to the right.} \end{array} \quad (4.39)$$

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

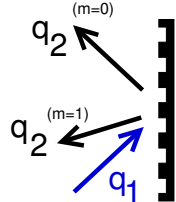
propagation through a free space of the length L and index of refraction n the beam parameter is transformed as follows:

$$M = \begin{pmatrix} 1 & \frac{L}{n} \\ 0 & 1 \end{pmatrix} \quad \xrightarrow{q_1} \boxed{\text{Space}} \xrightarrow{q_2} \quad (4.40)$$

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 m th 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:

$$M_t = \begin{pmatrix} M & 0 \\ -2/R_t & 1/M \end{pmatrix} \quad M_s = \begin{pmatrix} 1 & 0 \\ -2/R_s & 1 \end{pmatrix} \quad (4.41)$$


The diagram shows a vertical grating with a series of rectangular rulings. An incident beam, labeled q_1 in blue, strikes the grating from the bottom left at an angle. Two diffracted beams are shown: the zeroth-order beam, labeled $q_2^{(m=0)}$, and the first-order beam, labeled $q_2^{(m=1)}$, both pointing upwards and to the left.

with M given by:

$$M = \cos(\phi_m) / \cos(\alpha), \quad (4.42)$$

and effective radii as:

$$R_t = R \frac{\cos(\alpha) \cos(\phi_m)}{\cos(\alpha) + \cos(\phi_m)}, \quad (4.43)$$

$$R_s = \frac{2R}{\cos(\alpha) + \cos(\phi_m)}. \quad (4.44)$$

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: $q = i(2\text{ mm})^2\pi/\lambda$ (i.e. $w_0 = 2\text{ mm}$, $z_0 = 0\text{ mm}$).
- 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 components³.
- 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 it automatically recomputes 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 157 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:

$$E(t, z) = \sum_j a_j \exp(i(\omega_j t - k_j z)). \quad (4.45)$$

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

$$E(t, x, y, z) = \sum_j \sum_{n,m} a_{jnm} u_{nm}(x, y) \exp(i(\omega_j t - k_j z)). \quad (4.46)$$

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.

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.

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 it 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:

$$b_{\text{out}} = b_{\text{in}} \exp \left(i \Delta \omega n_r L / c - \left(\frac{1}{2} + n \right) \Psi_x + \left(\frac{1}{2} + m \right) \Psi_y \right), \quad (4.47)$$

(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 4.46 or Equation 4.1. In fact, the field amplitude is given in FINESSE (for one point in space, and $t = 0$) as:

$$E(t, x, y, z) = \sum_j \sum_{n,m} b_{jnm} u_n(x) u_m(y) \exp \left(-i \left(\frac{1}{2} + n \right) \Psi_t \right) \exp \left(-i \left(\frac{1}{2} + m \right) \Psi_s \right), \quad (4.48)$$

with

$$\Psi_t = \arctan \left(\frac{\text{Re} \{q_t\}}{\text{Im} \{q_t\}} \right), \quad \Psi_s = \arctan \left(\frac{\text{Re} \{q_s\}}{\text{Im} \{q_s\}} \right). \quad (4.49)$$

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 $\text{TEM}_{00}(q_1)$ 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 ($q_1(z_{\text{cav}}) \neq q_2(z_{\text{cav}})$), 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:

$$q_2 = \frac{q_1}{-2q_1/R_C + 1}, \quad (4.50)$$

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:

$$u_{nm}(q_1) \exp(i(\omega t - kz)) = \sum_{n', m'} k_{n, m, n', m'} u_{n' m'}(q_2) \exp(i(\omega t - kz')), \quad (4.51)$$

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 convolution [Bayer-Helms]:

$$k_{n, m, n', m'} = \exp(i 2kz' \sin^2(\frac{\gamma}{2})) \iint dx' dy' u_{n' m'} \exp(ikx' \sin \gamma) u_{nm}^*. \quad (4.52)$$

Since the Hermite-Gauss modes can be separated with respect to x and y , the coupling coefficients 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 4.8.

The convolution given in Equation 4.52 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:

$$K = \frac{1}{2}(K_0 + i K_2), \quad (4.53)$$

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 $q = i z_R + z - z_0$):

$$K = \frac{i(q - q')^*}{2 \operatorname{Im}(q')}. \quad (4.54)$$

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

$$k_{nn'} = (-1)^{n'} E^{(x)}(n!n'!)^{1/2} (1 + K_0)^{n/2+1/4} (1 + K^*)^{-(n+n'+1)/2} \{S_g - S_u\}, \quad (4.55)$$

where:

$$\begin{aligned}
 S_g &= \sum_{\mu=0}^{[n/2]} \sum_{\mu'=0}^{[n'/2]} \frac{(-1)^\mu \bar{X}^{n-2\mu} X^{n'-2\mu'}}{(n-2\mu)!(n'-2\mu')!} \sum_{\sigma=0}^{\min(\mu, \mu')} \frac{(-1)^\sigma \bar{F}^{\mu-\sigma} F^{\mu'-\sigma}}{a} (2\sigma)! (\mu - \sigma)! (\mu' - \sigma)!, \\
 S_u &= \sum_{\mu=0}^{[(n-1)/2]} \sum_{\mu'=0}^{[(n'-1)/2]} \frac{(-1)^\mu \bar{X}^{n-2\mu-1} X^{n'-2\mu'-1}}{(n-2\mu-1)!(n'-2\mu'-1)!} \sum_{\sigma=0}^{\min(\mu, \mu')} \frac{(-1)^\sigma \bar{F}^{\mu-\sigma} F^{\mu'-\sigma}}{(2\sigma+1)! (\mu - \sigma)! (\mu' - \sigma)!}.
 \end{aligned} \tag{4.56}$$

The respective formula for $k_{mm'}$ can be obtained by replacing the following parameters: $n \rightarrow m$, $n' \rightarrow m'$, $X, \bar{X} \rightarrow 0$ and $E^{(x)} \rightarrow 1$ (see below). The notation $[n/2]$ means:

$$\left[\frac{m}{2} \right] = \begin{cases} m/2 & \text{if } m \text{ is even,} \\ (m-1)/2 & \text{if } m \text{ is odd.} \end{cases} \tag{4.57}$$

The other abbreviations used in the above definition are:

$$\begin{aligned}
 \bar{X} &= (i z'_R - z') \sin(\gamma) / (\sqrt{1 + K^*} w_0), \\
 X &= (i z_R + z') \sin(\gamma) / (\sqrt{1 + K^*} w_0), \\
 F &= K / (2(1 + K_0)), \\
 \bar{F} &= K^* / 2, \\
 E^{(x)} &= \exp\left(-\frac{X\bar{X}}{2}\right).
 \end{aligned} \tag{4.58}$$

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:

$$k_{nmm'n'} = k_{nn'} k_{mm'}, \tag{4.59}$$

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

$$\begin{aligned}
 q &\rightarrow q_t \\
 \text{and} & \\
 w_0 &\rightarrow w_{t,0}, \text{ etc.,}
 \end{aligned} \tag{4.60}$$

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

$$\begin{aligned}
 n &\rightarrow m, \\
 n' &\rightarrow m', \\
 q &\rightarrow q_s, \\
 \text{thus} & \\
 w_0 &\rightarrow w_{s,0}, \text{ etc.,}
 \end{aligned} \tag{4.61}$$

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

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

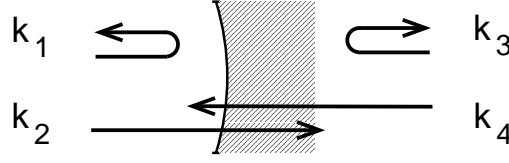


Figure 4.2: Coupling coefficients for Hermite-Gauss modes: for each optical element and each direction of propagation complex coefficients k for transmission and reflection have to be computed. In this figure k_1, k_2, k_3, k_4 each represents a matrix of coefficients $k_{nmn'm'}$ describing the coupling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

4.6.2 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.

Phase maps

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

$$E_{\text{refl}} = r \exp(i\varphi(x, y)) E_{\text{in}} \quad (4.62)$$

with

$$\varphi(x, y) = 2 \frac{n_1 \omega}{\lambda \omega_0} \phi_p(x, y) \quad (4.63)$$

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:

$$E_{\text{refl}} = r \exp(2i n_1 \phi_p(x, y)/\lambda_0) E_{\text{in}} \quad (4.64)$$

The transmission can be written as

$$E_{\text{trans}} = i t \exp(i (n_1 - n_2) \phi_p(x, y)/\lambda_0) E_{\text{in}} \quad (4.65)$$

Absorption maps

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

$$E_{\text{refl}} = r \sqrt{1 - L(x, y)} E_{\text{in}} \quad (4.66)$$

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

$$E_{\text{trans}} = i t \sqrt{1 - L(x, y)} E_{\text{in}} \quad (4.67)$$

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

$$E_{\text{refl}} = \sqrt{R(x, y)} \sqrt{1 - L} E_{\text{in}} \quad (4.68)$$

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

$$E_{\text{trans}} = i \sqrt{1 - R(x, y)} \sqrt{1 - L} E_{\text{in}} \quad (4.69)$$

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:

$$k_{n, m, n', m'} = \int \int dx' dy' u_{n', m'} \exp(2i n_1 \phi_p(x', y')/\lambda_0) u_{n, m}^* \quad (4.70)$$

which can be computed directly using a numerical integration routine.

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:

$$k_{n, m, n', m'}^{\text{map}} = \int \int dx' dy' u(n', m', q_1, n_1) A(x', y') u^*(n, m, q_1, n_1) \quad (4.71)$$

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 by a matrix multiplication:

$$k^{\text{total}} = k^{\text{other}} \times k^{\text{map}} \quad (4.72)$$

where k^{other} contains the standard coupling coefficients derived from mode mismatch or misalignment. This approach allows to seamlessly use together multiple maps as well as additional attributes to mirrors such as radius of curvature and alignment angle.

The following table⁴ 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.

<i>type of map</i>	<i>fields affected</i>	$A(x, y)$ (refl.)	$A(x, y)$ (trans.)
phase	reflection	$\exp(i 2k n_1 B(x, y))$	1
phase	transmission	1	$\exp(-i k B(x, y))$
phase	both	$\exp(i 2k n_1 B(x, y))$	$\exp(-i k (n_1 - n_2) B(x, y))$
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.

The map file format

A mirror map file can contain the mirror map as a grid $B(x, y)$ or as coupling coefficients $k_{n'm'nm}$, or both. 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): 100 100
% Step size (x,y): 0.0001 0.0001
% Scaling: 5.32e-07
```

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 $(\text{cols}-1)/2, (\text{rows}-1)/2$. Please note that this convention differs from those using a starting index of 1. In MATLAB, for example, the central element of a grid would be given by $(\text{cols}+1)/2, (\text{rows}+1)/2$ instead. **there is some inconsistency hidden**

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

here somewhere. In the matlab functions we don't do it this way. Maybe center need to be shifted by one point when saving.

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:

$$d_{\text{beam},x} = 2 w_x(z) \sqrt{\text{maxtem} + 0.5} \quad (4.73)$$

The maximum beam diameter is then computed as

$$d_{\text{beam}} = \max(d_{\text{beam},x}, d_{\text{beam},y}) \quad (4.74)$$

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 4.71. Note that in all cases the Gaussian beam parameter used to compute $u(n', m')$ and $u^*(n, m)$ are identical.

The integral is performed numerically by summing over all grid elements. The x, y coordinates used in the Hermite-Gauss functions u_{nm} are computed as follows:

$$\begin{aligned} x(i = 1 : \text{cols}) &= (i - x0) * xstep \\ y(j = 1 : \text{rows}) &= (j - y0) * ystep \end{aligned} \quad (4.75)$$

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).

Coupling coefficients can be stored similarly, starting with a header of twelve lines, for example:

```
% Coupling coefficients
% Name: test
% maxtem: 3
% qx1 (re im): 5 2.95262
% qy1 (re im): 5 2.95262
% qx2 (re im): 5 2.95262
```

```
% qy2 (re im): 5 2.95262
% nr1: 1
% nr2: 1
% lambda0: 1.064e-06
% angle: 0
% n1 m1 n2 m2    k11 k22 k12 k21 (real imaginary)
```

The first line indicates the start of a section storing coupling coefficients, the second line gives the name of the map from which the coefficients have been computed. The third line states the maximum order of Hermite-Gauss modes for which coefficients exist in this file. The lines four to seven give the Gaussian beam parameter at the front and back face of the mirror for which the coefficients have been computed. Similarly, line eight and nine contain the index of refraction at the front and back of the mirror surface. Line ten specifies the wavelength at which the coefficients have been derived, line eleven the rotation angle of the map and the twelfth line describes how the coefficients are stored in the following. This header is followed by the actual coefficients.

Surface map example: a focusing surface in transmission

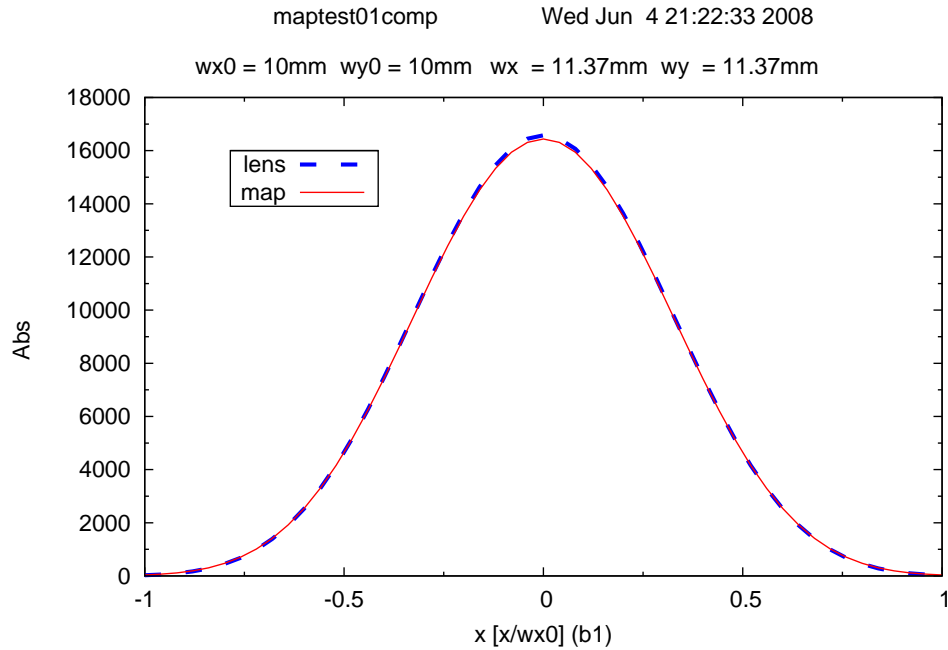


Figure 4.3: Beam shape comparison between setup with a lens component and a mirror surface map representing the same focal length

Test file provided by Jerome Degallaix. We create a transmission map with MATLAB which should mimic a lens:

```

Grid.num_points = 101; % The map size
Grid.origin_x = (Grid.num_points - 1)/2; % Center of the map in x axis
Grid.origin_y = (Grid.num_points - 1)/2; % Center of the map in y axis
Grid.size = 0.2; % Physical size of the grid in meter
Grid.resolution = Grid.size/(Grid.num_points-1); % Size of one pixel

Grid.vector = -Grid.size/2 + (0:Grid.num_points-1)*Grid.resolution;

% 2D matrix
[Grid.X,Grid.Y] = meshgrid(Grid.vector);
Grid.D2_square = Grid.X.^2 + Grid.Y.^2;

% Focal length of the converging 'lens' in meter
Focal_lens = 200;
% Optical path length induced is:
Delta_OPN = (-1/(2*Focal_lens))*Grid.D2_square;

maptype=0; % phase map
mapfield=2; % affect transmitted light only

map=FT_new_surface_map('mymap',Delta_OPN,maptype, mapfield, Grid.origin_x,
    Grid.origin_y,Grid.resolution, Grid.resolution,1);

FT_write_surface_map('mymap01.txt',map);

```

The MATLAB functions starting with 'FT' are from the collection *FinesseTools* which is available via the FINESSE download page, see also Section ??.

To run a FINESSE simulation using this map, the following input file can be used:

```

l i1 1 0 n1
gauss g1 i1 n1 1e-2 0

s s1 10 n1 n2
m m1 0 1 0 n2 n3
s s2 150 n3 n4

% load mirror map and apply to mirror m1
map m1 0 test mymap01.txt

% overwrite the original map file with new one,
% the second time this file is run the
% coefficients do not need to be created again
savemap m1 test mymap01.txt

% prints some debug information regarding the mirror maps
debug 64
% plot beam shape
beam b1 n4
xaxis b1 x lin -1 1 50
maxtem 8

```

To check this result we can run a different file which uses a lens component instead of the mirror with the surface map. For a proper comparison the same mode mismatch must be present. One problem when using mirror maps is that the beam tracing algorithm does not see them, which in this example results in a mode mismatch at the mirror. Thus we need to manually set the Gaussian beam parameter behind the lens to the value that is used at the front of the lens.

```
l i1 1 0 n1
gauss g1 i1 n1 10m 0

s s1 10 n1 n2
lens l1 200 n2 n3
s s2 150 n3 n4

gauss g2 l1 n3 10m 10
% plot beam shape
beam b1 n4
xaxis b1 x lin -1 1 50
phase 0
maxtem 8
```

Both results are shown in figure 4.3.

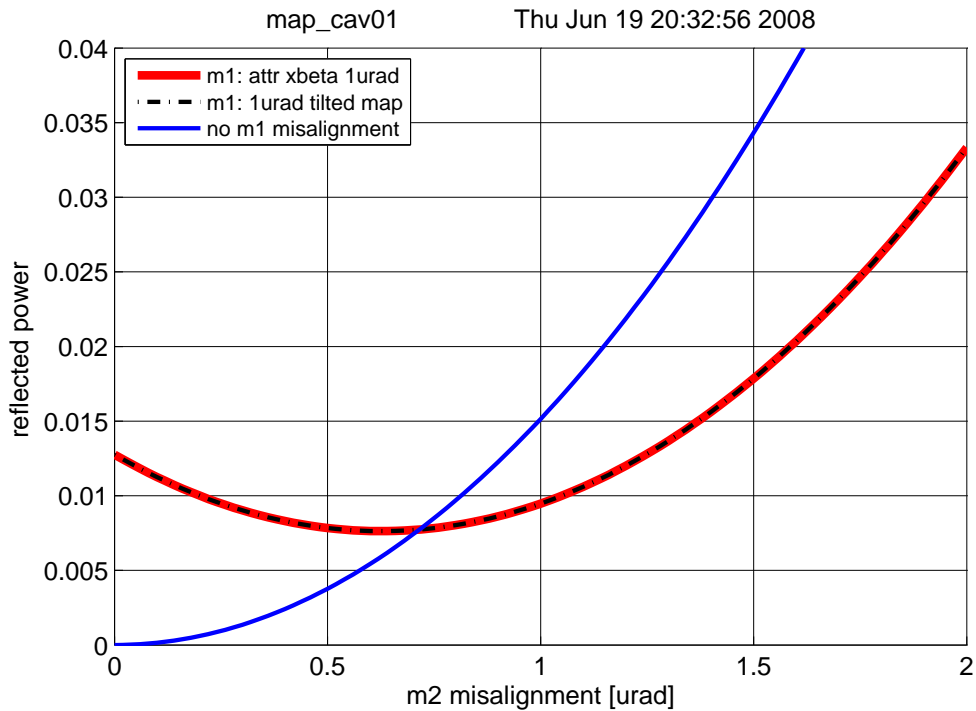


Figure 4.4: Comparison between a setup in which mirror *m1* is tilted by $1\ \mu\text{rad}$ using the *attr* command and the application of a phase map representing a $1\ \mu\text{rad}$ tilt.

Surface map example: a tilted mirror in reflection

Another simple example provided by Jerome Degallaix is a tilted mirror. The map can be created using MATLAB as before but with the following differences to the script:

```
% Define a uniform tilt
Tilt = 1E-6; % 1 urad
Delta_OPN = Grid.X*(Tilt);

% phase map in reflection
maptype=0;
mapfield=1;
```

We can then compare two FINESSE simulations: one in which this phase map is applied to a mirror and another in which no phase map is used but a mirror tilt is applied via the `attr xbeta` command. In order to maximize the effect, we use a cavity for this example:

```
l i1 1 0 n1

s s1 10 n1 n2

m m1 0.99 0.01 0 n2 n3
s s2 1k n3 n4
m m2 0.99 0.01 0 n4 n5
attr m2 Rc 2500

cav cav_mode m1 n3 m2 n4

pd0 Pref n2

xaxis m2 xbeta lin 0 2u 100
```

The coupling coefficients computed in the two cases will very likely have a different average phase so that we need to make sure the cavity is resonant. This can be achieved automatically by a locking system:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pseudo lock of cavity
ad ph_m11 0 n2*
ad ph_m12 0 n3*
noplot ph_m11
noplot ph_m12
set ph1 ph_m11 deg
set ph2 ph_m12 deg
func cphase = $ph2-$ph1-90
noplot cphase
lock clock $cphase 1m .1m
noplot clock
put m1 phi $clock
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The different tilts are achieved with:

```
attr m1 Rc 0 xbeta 1E-6
```

or:

```
% load mirror map and apply to mirror m1
% then save the new coupling coefficient
map m1 0 test mymap_tilt02.txt
savemap m1 test mymap_tilt02.txt
```

respectively. The results are shown in Figure 4.4.

4.6.3 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:

$$a'_{jn'm'} = \sum_{n,m} k_{n'm'nm} a_{jnm} \exp(i\varphi_{sb}), \quad (4.76)$$

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:

$$k_{nn'} = (-1)^{n'} E^{(x)}(n!n'!)^{1/2} \sum_{\mu=0}^{\min(n,n')} \frac{(-1)^\mu \bar{X}^{n-\mu} X^{n'-\mu}}{\mu!(n-\mu)!(n'-\mu)!}, \quad (4.77)$$

where

$$\bar{X} = X^* = \frac{-(z' - i z'_R) \sin \gamma}{w'_0}, \quad (4.78)$$

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 addition the exponential function can be neglected $E^{(y)} = 1, E^{(x)} = 1 + O(\gamma^2)$. Thus the sum reduces to:

$$\sum_{\mu=0}^{\min(n,n')} \frac{(-1)^\mu \bar{X}^{n-\mu} X^{n'-\mu}}{\mu!(n-\mu)!(n'-\mu)!} = O(X^2) + \begin{cases} \frac{(-1)^{n'} \bar{X}}{n'!} & \text{for } n = n' + 1 \\ \frac{(-1)^n X}{n!} & \text{for } n' = n + 1 \\ 1 & \text{for } n' = n \\ 0 & \text{otherwise} \end{cases} \quad (4.79)$$

The coupling coefficients can now be written as:

$$k_{nm'} = \begin{cases} \sqrt{n} \bar{X} & \text{for } n = n' + 1, \\ -\sqrt{n'} \bar{X}^* & \text{for } n' = n + 1, \\ 1 & \text{for } n' = n, \\ 0 & \text{otherwise.} \end{cases} \quad (4.80)$$

4.7 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.

4.7.1 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:

$$S = a \exp(i\Phi), \quad (4.81)$$

where

$$a = \sqrt{\sum_{n,m} |a_{n,m}|^2}, \quad (4.82)$$

$$\Phi = \text{phase} \left(\sum_{n,m} a_{n,m} \right). \quad (4.83)$$

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.

4.7.2 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:

$$S = \sum_{n,m} a_{nm} a_{nm}^*. \quad (4.84)$$

(as usual the fields referring to sidebands created by `fsg` 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:

$$S = \sum_{n,m} \sum_{n',m'} c_{nmn'm'} a_{nm} a_{n'm'}^*, \quad (4.85)$$

where c is the beat coefficient matrix. For example, a detector split in the x -direction is usually set up to detect the difference between the two halves thus being sensitive only to the beat signals of symmetric with asymmetric modes (in the x -direction). The coefficients are therefore:

$$c_{nmn'm'} = \begin{cases} 1 & \text{when } n + n' \text{ is odd and } m = m', \\ 0 & \text{otherwise.} \end{cases} \quad (4.86)$$

The subsequent demodulation of the signal is performed exactly as in plane-wave mode. Using split detectors the control signals for automatic alignment systems or other similar geometrical control systems can be calculated.

4.7.3 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:

$$s(x, y) = \sum_{ij} \sum_{nm} u_{nm}(x, y) u_{nm}^*(x, y) a_{inm} a_{jnm}^* \quad \text{with} \quad \{i, j \mid i, j \in \{0, \dots, N\} \wedge \omega_i = \omega_j\}. \quad (4.87)$$

If instead a frequency is specified the beam detector resembles an amplitude detector, it outputs the amplitude and the phase of the light field at the given frequency as a

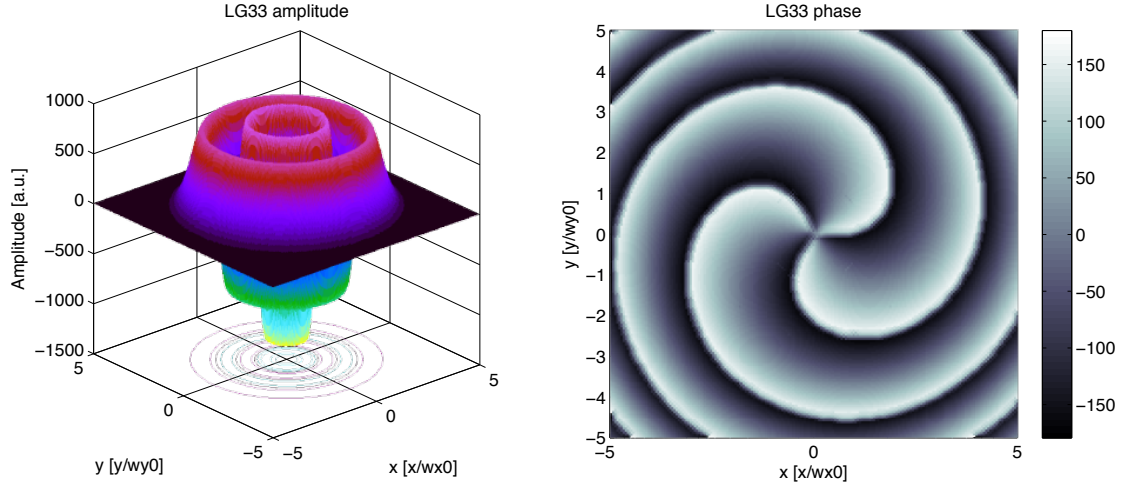


Figure 4.5: Example output for a beam detector: amplitude and phase map of a Laguerre Gauss LG33 mode. The data has been produced directly in FINESSE with a `beam ccd 0 n4` command and `yaxis abs:deg`. The phase unwrap and colouring has been done in MATLAB.

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:

$$z(x, y) = \sum_j \sum_{nm} u_{nm}(x, y) a_{jnm} \quad \text{with} \quad \{j \mid j \in \{0, \dots, N\} \wedge \omega_j = \omega_i\}. \quad (4.88)$$

4.8 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:

$$\kappa = \left(\frac{w_0}{2z_R} \right)^2 = \left(\frac{\lambda}{2\pi w_0} \right)^2. \quad (4.89)$$

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.

4.9 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.

4.9.1 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 integer 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.

4.9.2 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_{mn} 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.

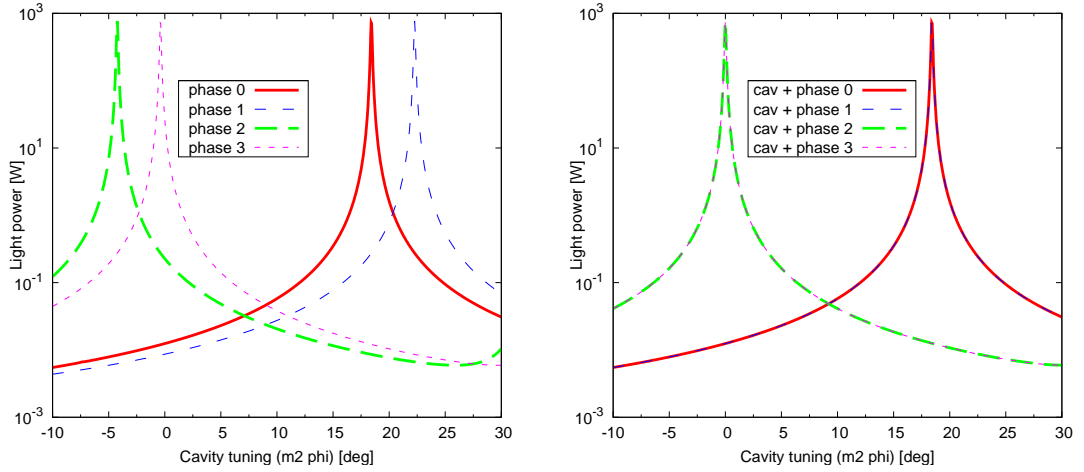


Figure 4.6: The light power in the cavity while one mirror is tuned around the cavity resonance. The left plot was created not using the `cav` command whereas the right plot shows the results obtained using `cav` and thus the cavity's eigenmodes. The different simplifications set by the `phase` command change the mirror tuning at which resonance is achieved. The respective offset values are listed in Table 4.1.

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 resonance (assuming a tuning of 0 degrees for the input mirror).

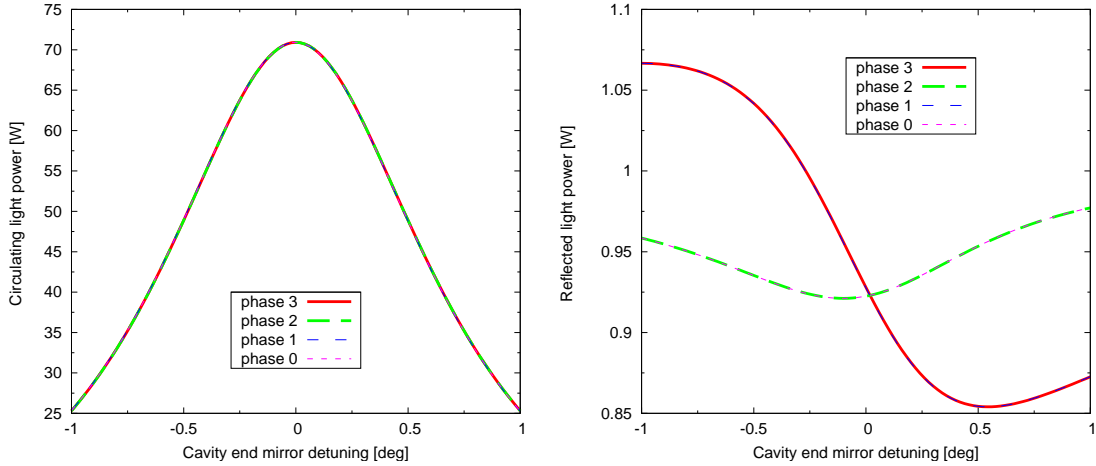


Figure 4.7: The two plots show the circulating power (left) and the reflected power (right) of a Fabry-Perot cavity for different `phase` settings. The input power is set to 1 W. The two phase settings which reset the phase of the coupling coefficients (`phase 3` and `phase 1`) can ‘create’ energy as 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.

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`.

4.9.3 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 the cavity eigenmodes.

Note that in this case the results do not depend on the setting of the `phase` command.

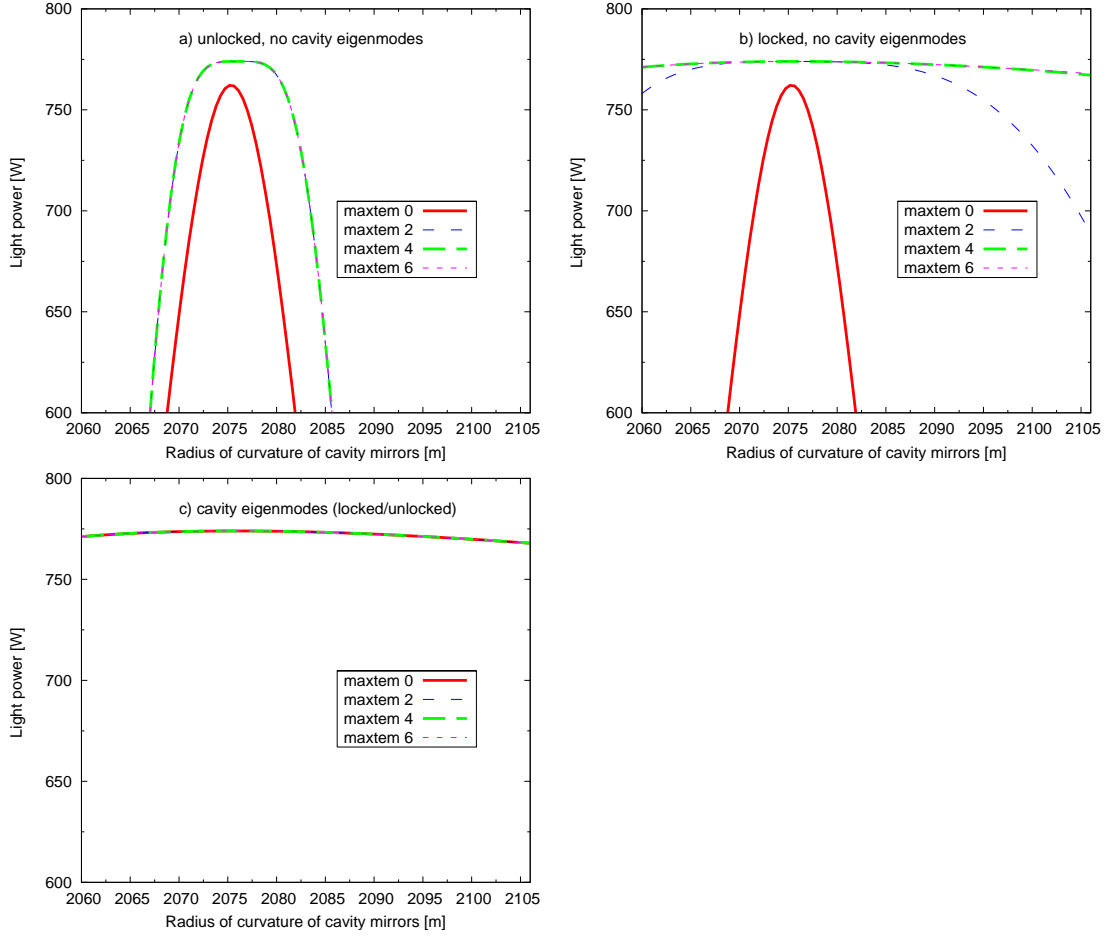


Figure 4.8: The three graphs compare the cavity power as a function of the radii of curvature of the cavity mirrors (the radii of curvature are changes symmetrically, i.e. $Rc_1 = -Rc_2$). The results do not change for different usage of the `phase` command. The input beam has been set to be matched to the eigenmodes of the cavity for $Rc = 2076$ m. Graph a) shows the situation when the `cav` command is not used and thus the system is analysed with a non-optimal 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.

4.10 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}_{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 $\vec{e}_{bs} = -\vec{e}_z$. Turning the beam splitter around the y -axis gives:

$$\vec{e}_{bs} = (\sin(\alpha), 0, -\cos(\alpha)). \quad (4.90)$$

Next, the beam splitter is tilted by the angle β_x around the x' -axis. Thus, the surface vector becomes:

$$\vec{e}_{bs} = (\sin(\alpha) \cos \beta_x, -\sin(\beta_x), -\cos(\alpha) \cos \beta_x). \quad (4.91)$$

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}_{bs} (see Figure 4.9):

$$\vec{a} = -(\vec{e}_z \cdot \vec{e}_{bs}) \vec{e}_{bs} = \cos(\beta_x) \cos(\alpha) \vec{e}_{bs}. \quad (4.92)$$

The reflected beam (\vec{e}_{out}) is then computed as:

$$\begin{aligned} \vec{e}_{out} &= -\vec{e}_z + 2(\vec{a} + \vec{e}_z) = 2\vec{a} + \vec{e}_z \\ &= (2 \cos^2(\beta_x) \cos(\alpha) \sin(\alpha), -2 \cos(\beta_x) \cos(\alpha) \sin(\beta_x), -2 \cos^2(\beta_x) \cos^2(\alpha) + 1) \\ &=: (x_o, y_o, z_o). \end{aligned} \quad (4.93)$$

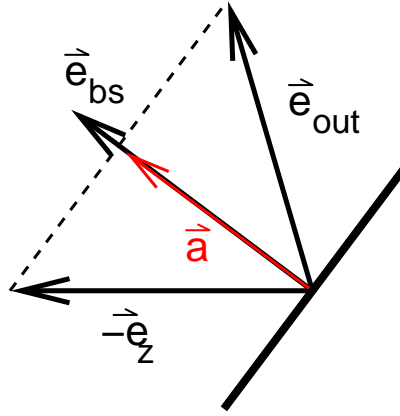


Figure 4.9: Mirroring of vector $-\vec{e}_z$ at the unit vector of the beam splitter surface \vec{e}_{bs} .

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}_{out} with the output vector for no tilt $\vec{e}_{out}|_{\beta_x=0}$. 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}_{out} and the x - z plane (γ_x). The latter is simply:

$$\sin(\gamma_x) = 2 \cos(\beta_x) \cos(\alpha) \sin(\beta_x) = \cos(\alpha) \sin(2\beta_x). \quad (4.94)$$

For small misalignment angles ($\sin(\beta_x) \approx \beta_x$ and $\sin(\gamma_x) \approx \gamma_x$), Equation 4.94 can be simplified to:

$$\gamma_x \approx 2\beta_x \cos(\alpha). \quad (4.95)$$

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:

$$\begin{aligned} \vec{e}_{out}|_{y_o=0} \cdot \vec{e}_{out}|_{\beta_x=0} &= \sqrt{x_o^2 + z_o^2} \cos(\gamma_y), \\ \Rightarrow \cos(\gamma_y) &= \frac{-1}{\sqrt{x_o^2 + z_o^2}} (x_o \sin(2\alpha) + z_o \cos(2\alpha)). \end{aligned} \quad (4.96)$$

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 $\alpha = 45^\circ$ and $\beta_x = 1$ 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 $\beta_x = 1^\circ$.

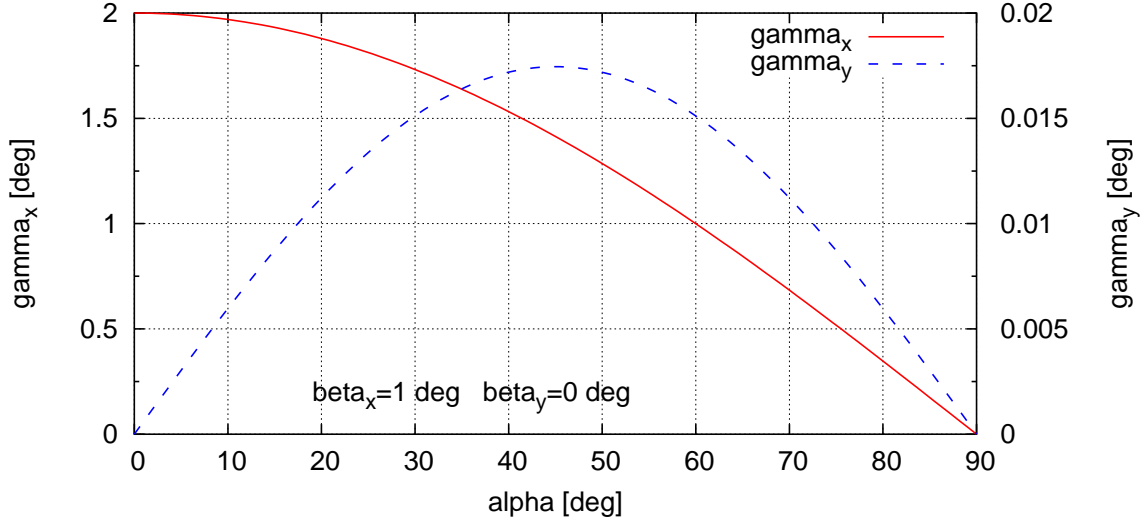


Figure 4.10: Misalignment angles of a beam reflected by a beam splitter as functions of the angle of incidence α . The beam splitter is misaligned by $\beta_x = 1^\circ$ and $\beta_y = 0$. Note that the values for β are very large, see Section 4.8, and have been chosen to artificially enlarge the coupling 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.

4.11 Aperture effects and diffraction losses

Current versions of FINESSE assume all optical components to have an infinite size transverse to the optical axis. Future versions might include some treatment of apertures using coupling coefficients as described in [VPB]. Currently losses or other diffraction effects due to small optical components cannot be simulated directly. The following calculation provides a very basic approximation for computing diffraction losses. In some simple cases the losses might be added as normal losses to the optical components.

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:

$$\begin{aligned}
 P_{\text{disk}} &= \int_{\text{disk}} I(r) = \int_0^{2\pi} d\phi \int_0^x dr r I(r) \\
 &= \frac{4P}{w} \int_0^x dr r e^{-2r^2/w^2} \\
 &= -P \int_0^x dr \partial_r e^{-2r^2/w^2} = -P \left[e^{-2r^2/w^2} \right]_0^x \\
 &= P \left(1 - e^{-2x^2/w^2} \right).
 \end{aligned} \tag{4.97}$$

A beam that is reflected at a mirror with diameter $2x$ will thus experience a power loss of at least:

$$P_{\text{loss}} = P e^{-2x^2/w^2}. \tag{4.98}$$

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 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.

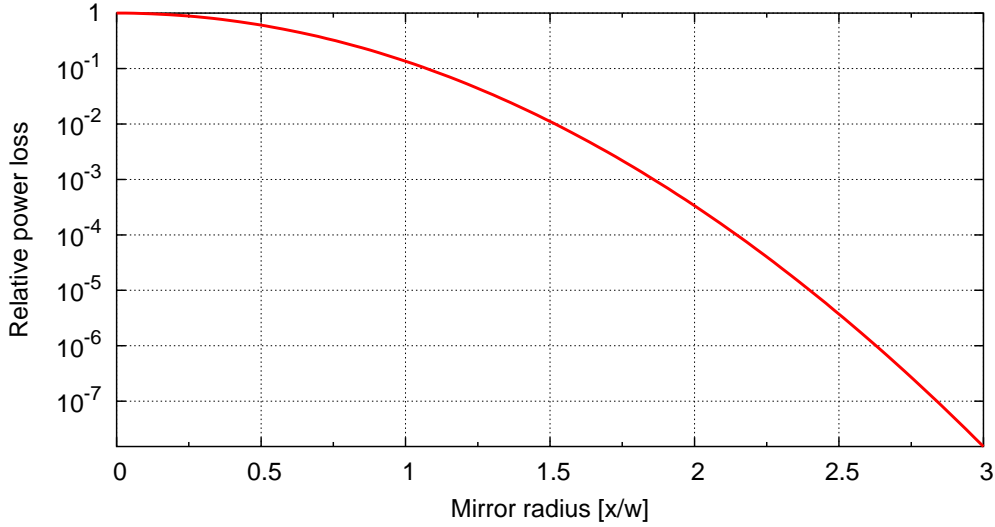


Figure 4.11: The plot shows a lower limit of the power loss experienced by a beam of size w on reflection at a mirror of diameter $2x$.

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.

Chapter 5

Advanced Usage

This chapter collects some thoughts and examples which probably do not concern the typical user but might be of interest to more advanced users.

5.1 FINESSE and MATLAB (Octave¹)

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 5.1.1.
- 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 5.1.2. 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.

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.

5.1.1 SimTools

Triggered by similar work of 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 (these are still available, see Section 5.1.3). 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*, that can be used to read, write, edit and execute FINESSE input files.

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 reflectivities
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.

5.1.2 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
```

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).

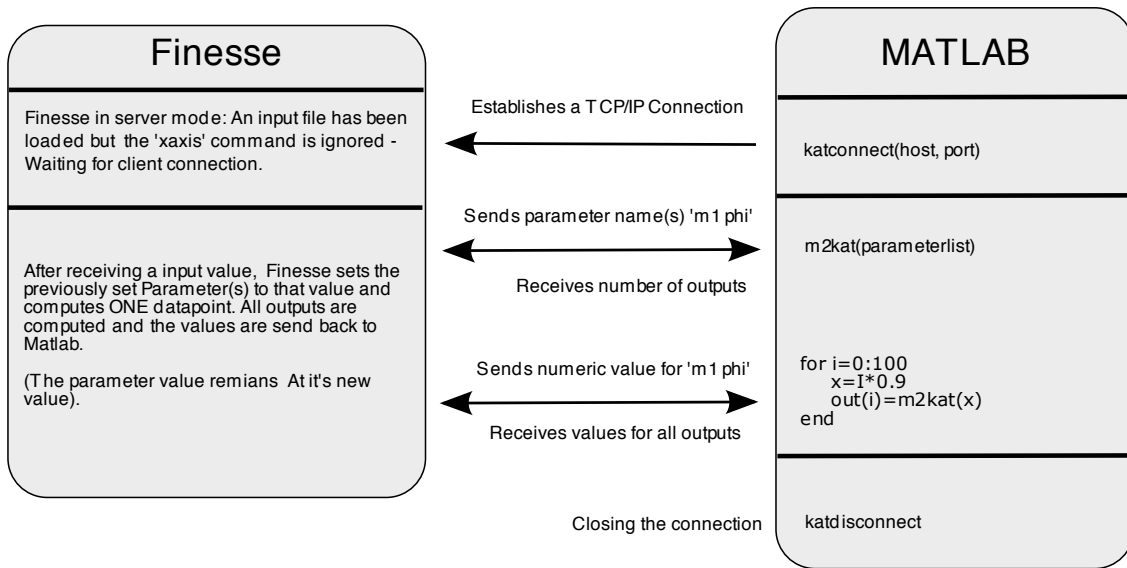


Figure 5.1: This sketch illustrates how the communication between Finesse and MATLAB/Octave would work in a simple example. FINESSE loads an input file and then becomes idle until a ‘katconnect’ command from a MATLAB session opens a TCP/IP connection. Then the ‘m2kat’ command can be used to send or receive 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 5.1. 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 command 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
%     repeatedly '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\b\b\n%3d%%',round(100*curr_call/ ...
                                                    max_calls)));
        last_print=0;
    end
end
end
end
end

disp(sprintf('\b\b\b\b\b\b\b\b\n100%% -- Complete!\n'));
toc
% send QUIT command and close socket
katdisconnect(socket);
% plot result
plot(x,out);

```

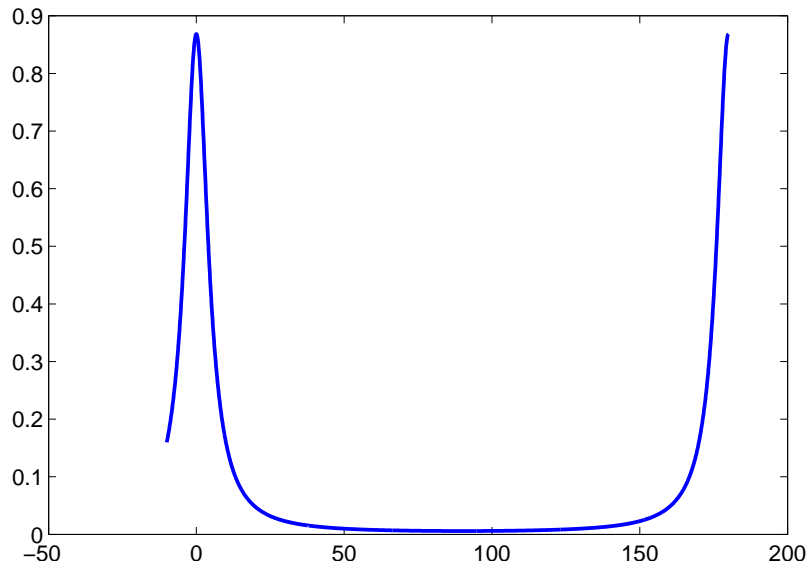


Figure 5.2: Graphical output of the example script demonstrating the communication between MATLAB and FINESSE running in server mode.

The terminal output of FINESSE during the example above would look similar to this:

```
moon:~/work/kat/test/m2kat$ kat --server 11000 cavity1.
```

```

-----
o_.-=.      FINESSE 0.99.7 (build 3227)
(\'".\|      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 5.2 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.

5.1.3 Octave example scripts

The following examples are collected in the file 'octave-examples-0.99.tar.gz' and the following description has been kindly provided by Gabriele Vajente.

The files in the archive contain some simple examples of how it is possible to use Octave to perform various analyses of FINESSE simulation results. These scripts have been tested with Octave 2.1.64 but should work also on previous versions and be compatible with MATLAB.

- `findzero.m`: contains a simple function that reads the output file of a FINESSE simulation and finds the zero crossing point of the signal. This function could be useful to find the locking point of a cavity using the Pound-Drever signal. For example you can run
`kat pdh-signal.kat`
and then give Octave the command
`findzero("pdh-signal.out")`

- `resort.m`: this function reads data from an output file created by tuning two parameters at a time. It returns a vector of x values, a vector of y values and the signal values arranged on a square grid. This can be used to plot the results within Octave. For example, you can run
`kat 3D.kat`
and then give in Octave the following commands
`[x,y,sig] = resort("3D.out",40)`
`mesh(x,y,sig)`
to have a 3D plot of the simulation results.
- `multifindzero.m`: this script reads the values of two signals as a function of two tunings and searches for a point where both signals are zero. This is useful to simulate the behaviour of an optical system with two control loops that tries to keep two control signals to zero. For example, you can use the file `multi.out` which contains two signals as a function of two parameters, and issue in Octave the command
`multifindzero`
Note that when one of the two signals does not cross zero in a given range of parameters, then the script can still find a false zero. This case is simply identified by looking at the plot made by the script itself.
- `search.m`: this is quite a complex example, which shows among other things a possible way to call FINESSE from an Octave script. This example show an iterative procedure to find the locking point of the VIRGO interferometer, simulating the behaviour of four control loops that tries to keep four corresponding signals to zero. This script also uses the files `search.kat`, `NE.temp`, `WE.temp`, `BS.temp`, `PR.temp` as starting points to build a correct FINESSE input file for very step.

For questions and comments please contact Gabriele Vajente vajente@sns.it.

5.2 Speed improvements

The time needed for a simulation with FINESSE is of course strongly connected with the simulation task. In most cases FINESSE will only take seconds to produce a result in which case a speed optimisation is not required. However, with some more complex problems, for example, when a large number of higher-order modes are used, the computation time can increase dramatically. The sections below give some information and general advise on how to avoid long computation times.

5.2.1 Higher-order modes

The most dramatic change is computation speed can be seen observed when the number of higher order modes is changed with the `maxtem` command. Figure 5.3 shows some measured computation times for example simulations as a function of the value given with `maxtem`. In general, the computational time scales like m^4 or m^5 with m being the

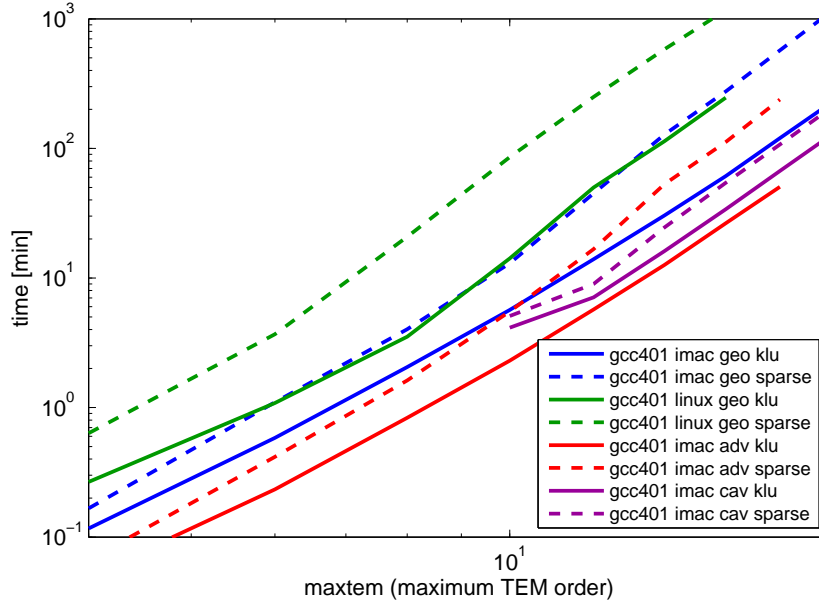


Figure 5.3: The graph shows measured computation times for three example optical setups: ‘geo’ represents the optical configuration of GEO 600, ‘adv’ refers to Advanced Virgo and ‘cav’ uses a simple two-mirror cavity. All versions of FINESSE tested here have been compiled with the gcc compiler version 4.0.1. The test runs have been performed on an iMac or a PC running linux. The terms ‘klu’ and ‘sparse’ refer the options ‘-sparse’ and ‘-klu’, see text. The scaling is slightly different for the different files and computers. However, generally the computational time scales like m^4 or m^5 with m being the maximum TEM order given via `maxtem`.

maximum TEM order. In consequence large values for `maxtem` should be used only in special cases when a high spatial resolution of the beam is required.

5.2.2 The `lock` command

The `lock` command as described in the [Syntax reference](#) and in the example files given in [A.1.7](#), can be used to model 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 order to perform the iteration. Used correctly the `lock` command can be an effective tool. However, it is important to not use it unnecessarily as even a simple `lock` often increases the computation time by factors of five.

Tuning the `lock` If you use a `lock` you should carefully tune 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.

5.2.3 KLU versus SPARSE

FINESSE currently implements two different sparse matrix solvers, the SPARSE 1.3 package [\[Sparse\]](#) and the KLU library [\[KLU\]](#). In my experience the KLU package provides a better performance for very large interferometer matrices. In particular, when many higher-order modes are used KLU typically gives times which are lower by a factor $\maxtem/3.5$, depending on the computer as well as the input file, see, for example [Figure 5.4](#). For settings of `maxtem`= 3 the SPARSE package can give better results, up to a factor of two faster.

Therefore currently FINESSE automatically switches to KLU when `maxtem` is set to values of 4 or higher. Otherwise the SPARSE package is used by default. This default behaviour can be changed manually with the options `'-klu'` and `'-sparse'` which switch to the respective package regardless of the `maxtem` value.

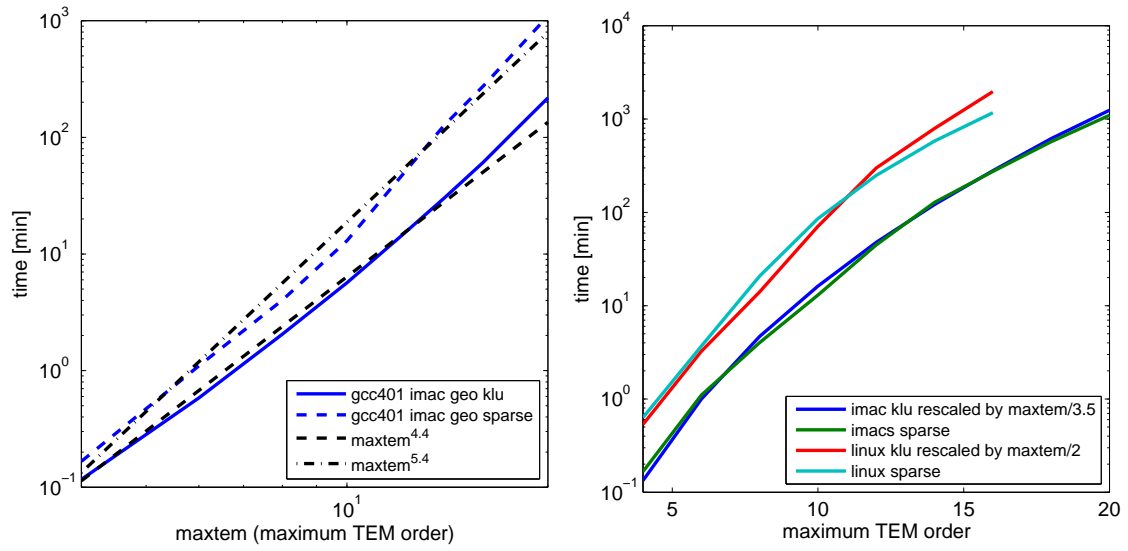


Figure 5.4: The left plot shows two traces from Figure 5.3 plotted with some power laws. The right plot compares the timing for the ‘klu’ and the ‘sparse’ options.

Appendix A

Example files

In this chapter we give some examples for using FINESSE. The first section shows a set of simple standard input files. The second section gives an example of how to use FINESSE together with Perl. A Perl pre-processor ‘mkat’ is part of the FINESSE distribution. The third section shows how FINESSE can be used together with Matlab or Octave.

For the advanced user it might also be helpful to study more complex examples. I provide the input files I created for the GEO 600 and VIRGO detectors on the web.

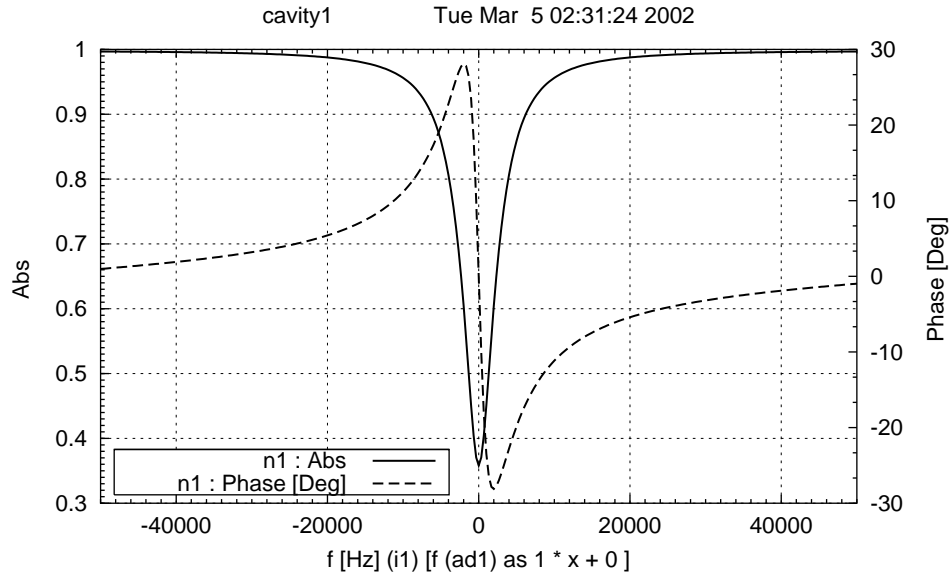
In addition, a VIRGO note [Freise04] gives a step by step explanation of the simulation for the commissioning of the VIRGO north arm cavity. All input files used for this work are currently available at

<http://wwwcascina.virgo.infn.it/alignment/simulation/finesse/files/northarm>.

A.1 Simple examples

The following sections provide the input file and the resulting plots of several simple example simulations. The example input files are identical to those provided as part of the FINESSE package.

A.1.1 cavity1.kat



```
#-----
# cavity1.kat , example file for kat 0.99
#
# freise@rzg.mpg.de 01.03.2005
#
# Amplitude and phase of the light reflected by the cavity.
#
# The interferometer :
#
# Fabry-Perot Cavity
#
#           m1                               m2
#           .-.                               .-.
#           ||                               ||
# --->  n1  ||  n2  . . . . . n3  ||
#           ||                               ||
#           ||                               ||
#           '._                               '._
#-----

m m1 0.9 0.1 0 n1 n2      # mirror R=0.9 T=0.1 phi=0
s s1 1200 n2 n3          # space L=1200
m m2 0.8 0.2 0 n3 dump    # mirror R=0.8 T=0.2 phi=0

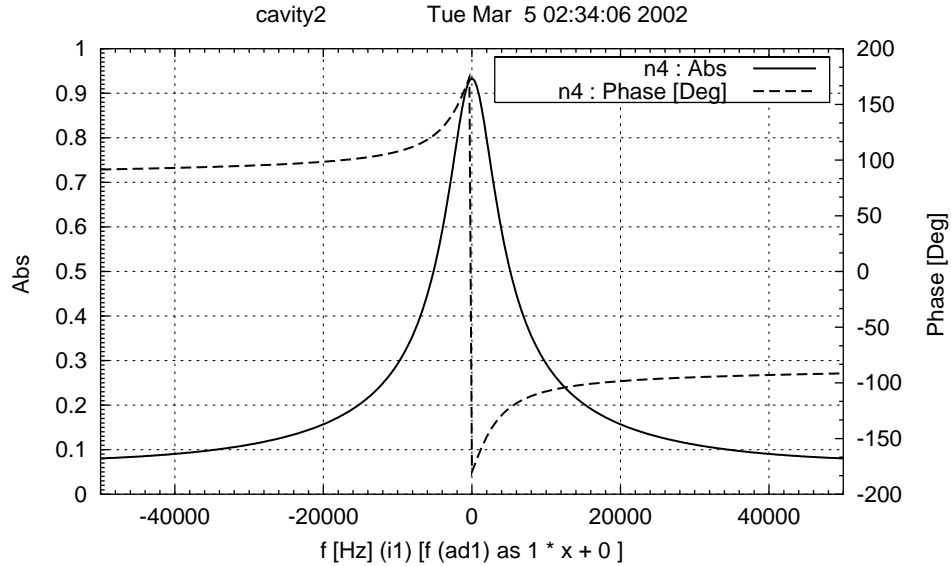
l i1 1 0 n1              # laser P=1W f_offset=0Hz

ad ad1 0 n1              # amplitude detector f=0Hz

xaxis i1 f lin -50k 50k 300  # x-axis : frequency of input light
```

```
put ad1 f $x1                                # from -50k to 50k (300 steps)
#syntax for kat 0.98 and before:              # put xaxis also to detector frequency
#xparam ad1 f 1 0
yaxis abs:deg                                # y-axis : Intensity and Phase
```

A.1.2 cavity2.kat



```
#-----
# cavity2.kat , example file for kat 0.99
#
# freise@rzg.mpg.de 01.03.2005
#
# Amplitude and phase of the light transmitted through the cavity.
#
# The interferometer :
#
# Fabry-Perot Cavity
#
#           m1                               m2
#           .-.                               .-.
#           ||                               ||
# ---->  n1 || n2 . . . . . s1 . . . . . n3 || n4 ---->
#           ||                               ||
#           ||                               ||
#           ',                               ',
#-----

m m1 0.9 0.1 0 n1 n2      # mirror R=0.9 T=0.1 phi=0
s s1 1200 n2 n3           # space L=1200
m m2 0.8 0.2 0 n3 n4      # mirror R=0.8 T=0.2 phi=0

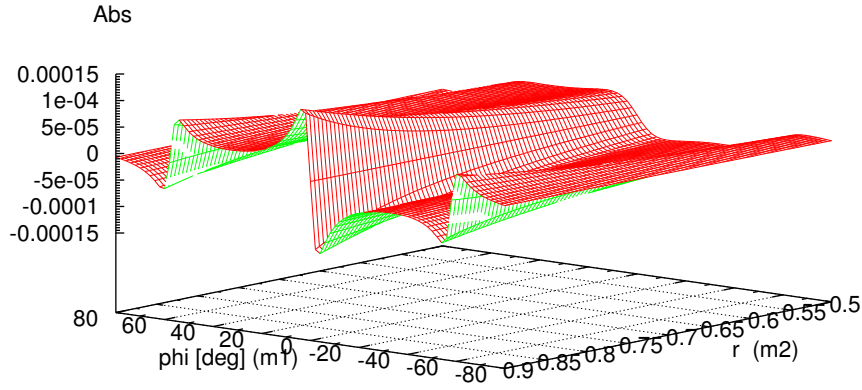
l i1 1 0 n1              # laser P=1W f_offset=0Hz

ad ad1 0 n4              # amplitude detector f=0Hz

xaxis i1 f lin -50k 50k 300  # x-axis : frequency of input light
```

```
put ad1 f $x1                # from -50k to 50k (300 steps)
#syntax for kat 0.98 and before
#xparam ad1 f 1 0
yaxis abs:deg                # y-axis : Intensity and Phase
```

A.1.3 3D.kat



inphase : ———

```
#-----
# 3D.kat test file for kat 0.99
# (Pound-Drever-Hall in 3D)
#
# freise@rzg.mpg.de 02.03.2005
#
#
#               m1               m2
#       .-----.   .-.   .-.
#       |      |   | |   | |
# --> n0 | EOM | n1 | | n2 . . . . . n3 | |
#       |      |   | |   | |
#       '-----'   | |   | |
#                   '._'   '._'
#-----

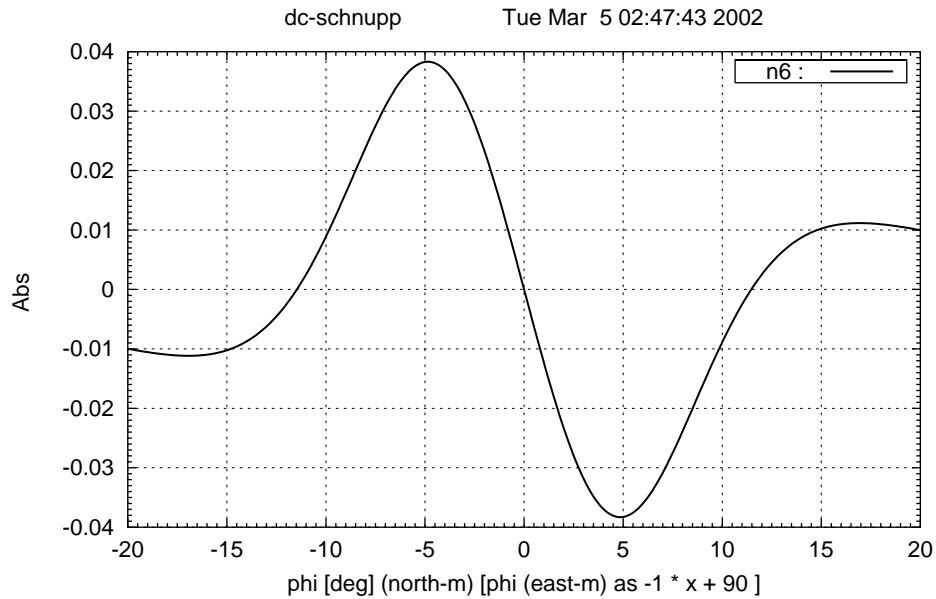
# a Fabry-Perot cavity
m m1 0.9 0.0001 0 n1 n2      # mirror R=0.9 T=0.0001 phi=0
s s1 1200 n2 n3              # space L=1200
m m2 0.8 0.1 0 n3 dump      # mirror R=0.8 T=0.1 phi=0

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 detecor + mixer
                               # f_demod=40kHz phase=0
xaxis m1 phi lin -80 80 80   # xaxis: phi at mirror m1
                               # from -80 to 80 (80 steps)
x2axis m2 r lin 0.5 0.9 80   # second xaxis (for 3D plot)
```

```
func t = 1.0-$x2          # r at mirror m2, 0.5 to 0.9
put m2 t $t              # compute t=1-r
#syntax for kat 0.98 and before
#x2param m2 t -1 1       # set new transmission of m2
yaxis abs                 # yaxis: plot 'as is'

GNUPLOT                  # some Gnuplot settings for
set view 70, 220, ,      # the 3D plot ...
set hidden3d
END
```

A.1.4 dc-schnupp.kat



```
#-----
# dc-schnupp.kat test file for kat 0.99
#
# freise@rzg.mpg.de 02.03.2005
#
# The "#" is used for comment lines.
#
# Power recycled Michelson Interferometer
#       with Schnupp Modulation :
#
#
#
#               <----->  north-m
#
#               n7
#
#               north-s
#
#               bs1          east-m
#               n4 .>'
#               .>'
# n0 |EOM| n1 | | n2 west-s n3 .>' n5 east-s n8 | |
#   '----'   | |               <.>'
#             | |
#             | |
#
#               n6
#
#               |
#               |
#               |
```



```
#                                     V
#
#                                     photo detectors
#-----

m north-m 1 0 0 n7 dump             # north end mirror
m east-m 1 0 90 n8 dump             # east end mirror
m west-m 0.9 0.1 0 n1 n2           # power recycling mirror

s north-s 1201 n4 n7                # north arm
s east-s 1199 n5 n8                # east arm
s west-s 1 n2 n3                   # west arm

bs bs1 0.5 0.5 0 45 n3 n4 n5 n6    # beam splitter (50/50)

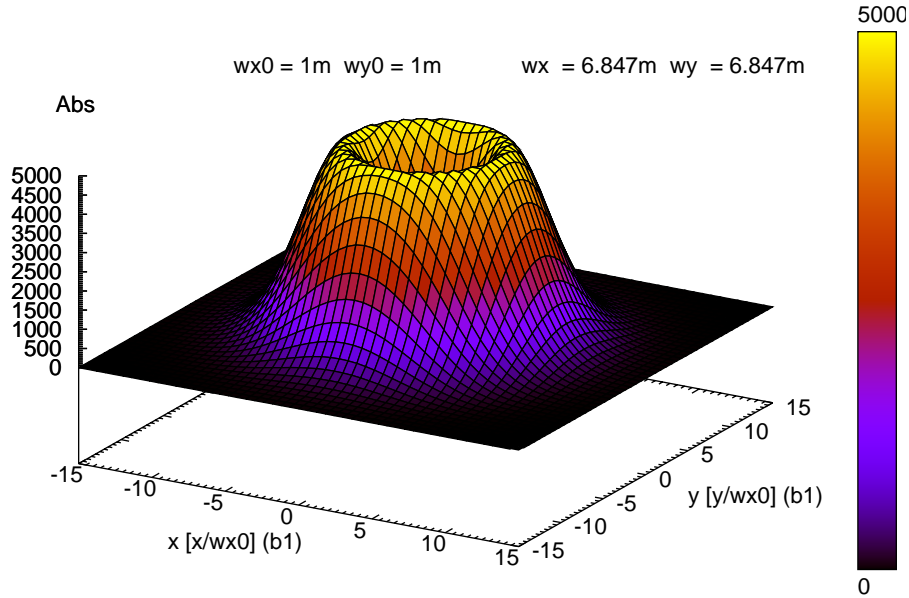
l i1 1 0 n0                        # laser 1W

mod eo1 1M 0.45 1 pm n0 n1         # Schnupp (in-line) modulation

pd1 out1 1M 50 n6                  # output photodiode + mixer

# tuning the end mirrors in differential mode
xaxis north-m phi lin -20 20 1000
func dm = 90-$x1
noplot dm
put east-m phi $dm
#syntax for kat 0.98 and before
#xparam east-m phi -1 90
yaxis abs
```

A.1.5 donut.kat



```
# FINESSE example file for testing the Hermite-Gauss extension
# adf 020402
#
# donut-mode example
#
# simply plots a nice donut mode
#

maxtem 2                # maximum TEM order = 2
trace 6                 # verbose mode

l i1 1 0 n1             # input beam with 1 Watt power

# no specify the relative power factors for the modes
# of the input i1:

tem i1 0 0 0 0          # nothing in TEM 00
tem i1 0 1 1 0          # 1 in TEM 01
tem i1 1 0 1 90         # and also 1 in TEM 10, with a phase
                        # offset of 90 degrees

s s1 10 n1 n2           # a 10 meters long space

gauss g1 i1 n1 0.001 10 # setting a Gaussian parameter

beam b1 n2              # detecting the beam behind the space

# for the 2-dimensional cross section the xaxis and x2axis are
```

```
# set to the x and y coordinates of the beam analyser:
```

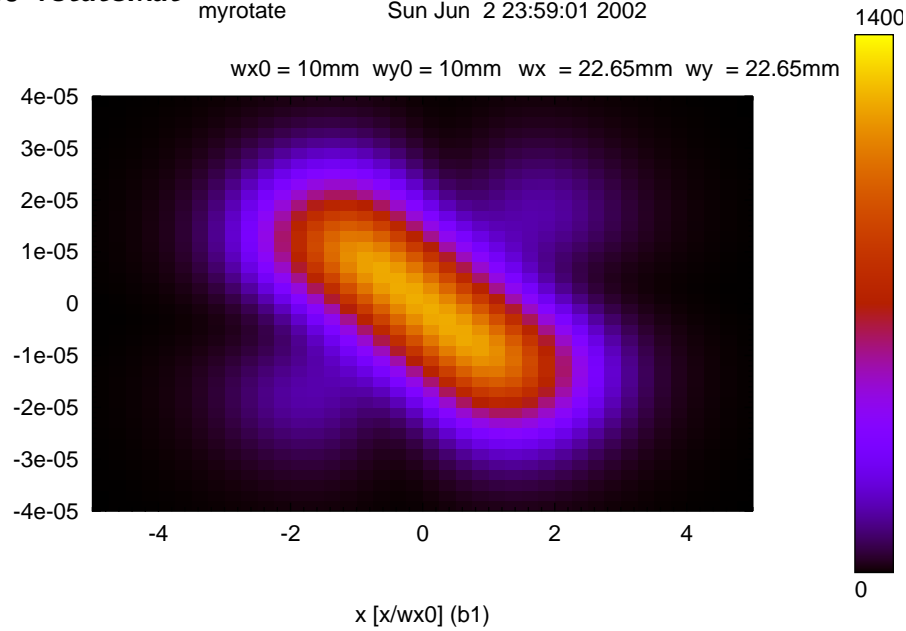
```
xaxis  b1 x lin -15 15 50
```

```
x2axis b1 y lin -15 15 50
```

A.1.6 rotate.kat

myrotate

Sun Jun 2 23:59:01 2002



```
# FINESSE example file for testing the Hermite-Gauss extension
# adf 020402
#
# rotating mirror example
#
# plots the effect of a beam splitter rotation:
# one can see how the 'linear range' for the
# pointing as a function of beta depends
# on the maximum order (maxtem).
```

```
maxtem 1
```

```
trace 6
```

```
l i1 1 0 n1
```

```
s s1 100 1 n1 n2
```

```
# a beam splitter as a turning mirror:
```

```
bs bs1 1 0 0 0 n2 n3 dump dump
```

```
# setting the misalignment angle xbeta to zero
```

```
attr bs1 xbeta 0.0
```

```
s s3 500 1 n3 n4
```

```
gauss g1 bs1 n3 0.01 100
```

```
beam b1 n4
```

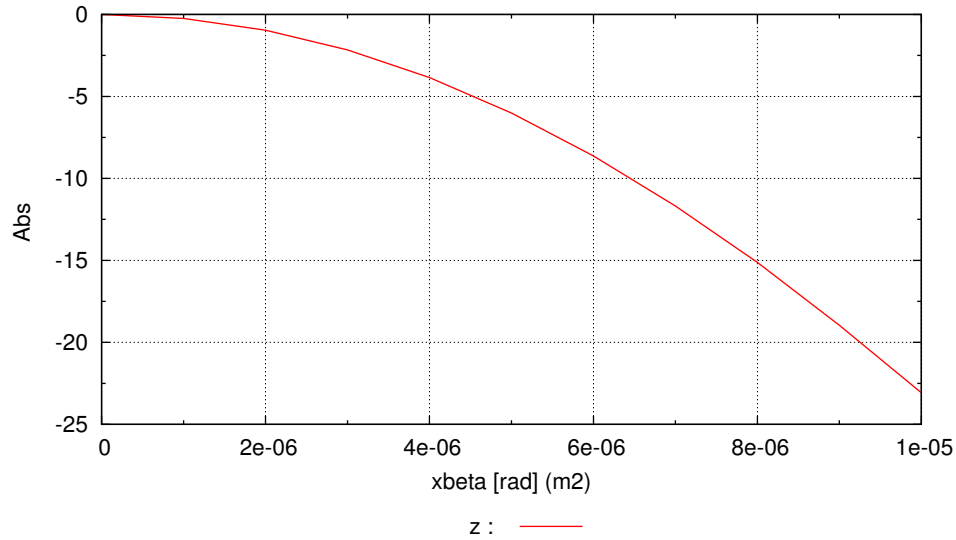
```
# The xaxis is set to the x position of the beam analyser.
```

```
# The 1-dimensional cross section of the beam is plotted.  
# The second axis tunes the alignment angle of bs1  
  
xaxis b1 x lin -5 5 40  
x2axis bs1 xbeta lin -40u 40u 40  
  
GNUPLOT  
set view 36, 44  
set contour  
END
```

A.1.7 lock-cav1.kat

lock-cav1

Sun Mar 6 00:32:28 2005



```
#-----
# pdh_signal.kat test file for kat 0.99
# (Error signal of the Pound-Drever-Hall signal)
#
# freise@aei.mpg.de 07.02.2005
#
#
#
#               m1               m2
#               .-.             .-.
#               | |             | |
# --> n0  | EOM | n1 | | n2  . . . . . n3 | |
#               | |             | |
#               '-----'       | |
#               | |             | |
#               '._'             '._'
#-----

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 0.9 0.1 0 n3 n4         # mirror R=0.9 T=0.1 phi=0
attr m2 Rc 1400              # ROC for m2 = 1400m

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

cav cavity1 m1 n2 m2 n3      # compute cavity eigenmodes
maxtem 3                     # TEM modes up to n+m=3
time                          # print computation time
```

```
##### locking the cavity #####
#
# The cavity is locked with the Pound-Drever-Hall
# technique, the feedback is connected to m1.
# The resulting plot contains only the feedback
# signal of the control loop.

pd1 pdh 40k 0 n1          # diode for PDH signal
noplplot pdh              # do not plot this output

# define error signal 'err'= Re(photo diode output)
set err pdh re
# define control loop:
# output : feedback signal = $z
# input  : error signal = $err
# parameters:
# - loop gain = -10000
# - 'accuracy' threshold for error signal = 100n
lock z $err -10k 100n

# connect the feedback to the input mirror:
put* m1 phi $z            # add $z to m1 tuning

#####

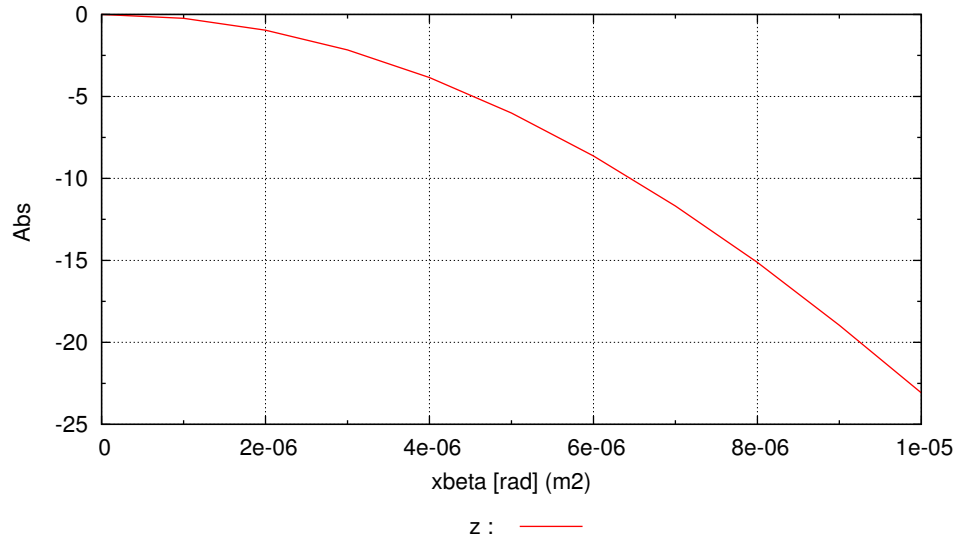
xaxis m2 xbeta lin 0 10u 10    # tune the angle of m2

yaxis abs
gnuterm x11
pause
```

A.1.8 lock-cav2.kat

lock-cav2

Sun Mar 6 00:32:33 2005



```
#-----
# lock-cav2.kat test file for kat 0.99
# based on lock-cav1.kat test file for kat 0.99
#
# Example for locking a Fabry-Perot cavity with FINESSE
# with a normalised error signal
#
# freise@aei.mpg.de 06.03.2005
#
#
#               m1               m2
#       .-----.   .-.   .-.
#       |      |   | |   | |
# --> n0 | EOM | n1 | |   n2 . . . . . n3 | |
#       |      |   | |   | |
#       '-----'   | |   | |
#                   '._'   '._'
#-----

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 0.9 0.1 0 n3 n4        # mirror R=0.9 T=0.1 phi=0
attr m2 Rc 1400              # ROC for m2 = 1400m

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
```



```
cav cavity1 m1 n2 m2 n3      # compute cavity eigenmodes
maxtem 3                     # TEM modes up to n+m=3
time                          # print computation time

##### locking the cavity #####
#
# The cavity is locked with the Pound-Drever-Hall
# technique, the feedback is connected to m1.
# The resulting plot contains only the feedback
# signal of the control loop.

pd1 pdh 40k 0 n1             # diode for PDH signal
noplot pdh                   # do not plot this output

pd power n4                  # transmitted power
noplot power                 # do not plot this output

# define error signal 'err'= Re(photo diode output)
set err pdh re
# define normalisation signal 'p' = transmitted power
set p power re
# compute y = normalised error signal
func y = $err/(1000*$p+0.0000001)
noplot y                     # do not plot this output
# define control loop:
# output : feedback signal = $z
# input  : error signal = $y
# parameters:
# - loop gain = -10000
# - 'accuracy' threshold for error signal = 100n
lock z $y -10k 100n

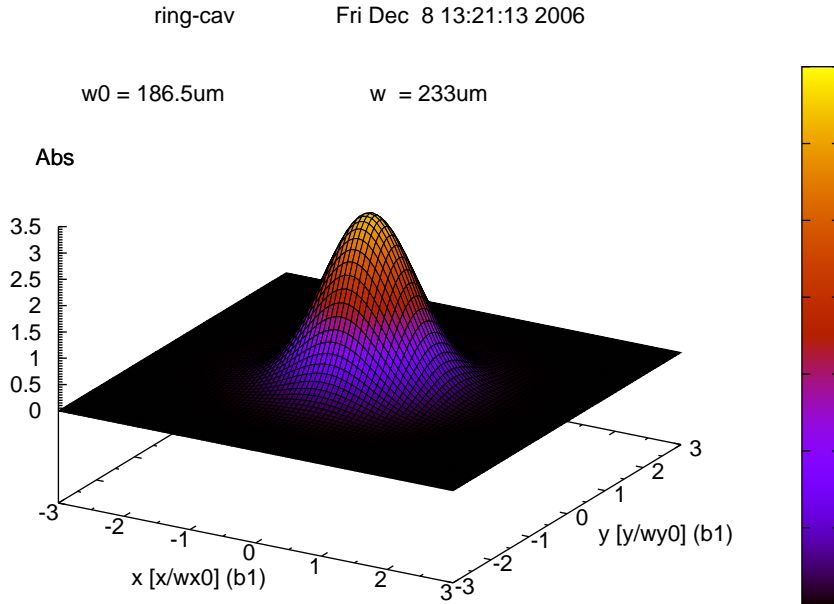
# connect the feedback to the input mirror:
put* m1 phi $z               # add $z to the m1 tuning

#
#####

xaxis m2 xbeta lin 0 10u 10  # tune the angle of m2

yaxis abs
gnuterm x11
pause
```

A.1.9 ring-cav.kat



```

#-----
# ring-cav.kat test file for kat 0.99
# triangular ring cavity as a mode cleaner
#
# adf@rzg.mpg.de 29.11.2006
#
#
#
#      MMC2b
#
#      .---,
# nn2 | /  n5      ,---,
#      | '  ,---,
# <-- | | +
#      | .  '---...
#      | \      '---...
#      '---+ n3      '---...
#
#
#
#
#      ,-.
#      (  \  MMC2c
#      |   \
#      |   \ n6
#      |   \ ,---,
#      |   \ n7
#      |   \
#      |   \
#      |   \
#      |   \ n8
#      |   \ /  MMC2a
#      |   \ /  n2
#      |   \ /  <-----
#      |   \ /  n1
#      |   \
#      |   \
#
maxtem 1

# defining an 'ugly' input mode to

```

```
# demonstrate the mode cleaning effect
l i1 1 0 n0
tem i1 0 0 1 40          # TEM 00
tem i1 0 1 1 0          # TEM 01
tem i1 1 0 1 90          # TEM 10

s s1 1 n0 n1

# input coupler
bs1 MMC2a 1532u 67u 0 0 n1 nrefl n2 n8

s sAB 3.9757 n2 n3

bs1 MMC2bi 1360u 58u 0 0 n3 n5 n4 dump
attr MMC2bi Rc 6.72

s sBC 3.9757 n5 n6

bs1 MMC2c 114u 62u 0 0 n6 n7 dump dump

s sCA 0.15 n7 n8

# The substrate of spherical output coupler
# serves as a lense for a mode matching telescope
s sMMC2b 0.05 1.44963 n4 nn1
m1 MMC2bo 1 0 0 nn1 nn2
attr MMC2bo Rc 0.35

s send 1 nn2 nn3 # this is the output node

cav c1 MMC2bi n5 MMC2bi n3

# You can see the mode cleaning effect by
# comparing the incident beam with the
# transmitted beam:
# profile of the incident beam
#beam b1 n0*
# profile of the transmitted beam
beam b1 nn3
# profile of the reflected beam
#beam b1 nrefl

xaxis b1 x lin -3 3 80
x2axis b1 y lin -3 3 80

gnuterm x11
pause

GNUPLOT
set pm3d at s hidden3d 100 solid
```

```
set nosurface
unset hidden3d
set colorbox v
set colorbox user origin .95,.1 size .04,.8
set style line 100 lt -1 lw 0
set palette rgbformulae 7,5,15
unset grid
END
```

A.2 mkat - a Perl preprocessor for Finesse

‘mkat’ is a Perl script which serves as a preprocessor for FINESSE¹. It automates repetitive calls of ‘kat’. Using this preprocessor you can do several calculations at once by using only one input file. ‘mkat’ reads the input file and looks for lines starting with a ‘tag’. A tag consists of a name, a number, and a colon, e.g. ‘run1:’. If ‘mkat’ finds one tag name with different numbers it creates new input files separating different tag numbers. For example an input file with:

```
...
m m1 .9 .1 0 n1 n2
m m2 .9 .1 0 n4 n5
run1:xaxis m1 phi lin 0 90 100
run2:xaxis m2 phi lin 0 90 100
```

would be separated into two files:

File1:

```
...
m m1 .9 .1 0 n1 n2
m m2 .9 .1 0 n4 n5
xaxis m1 phi lin 0 90 100
```

and File2:

```
...
m m1 .9 .1 0 n1 n2
m m2 .9 .1 0 n4 n5
xaxis m2 phi lin 0 90 100
```

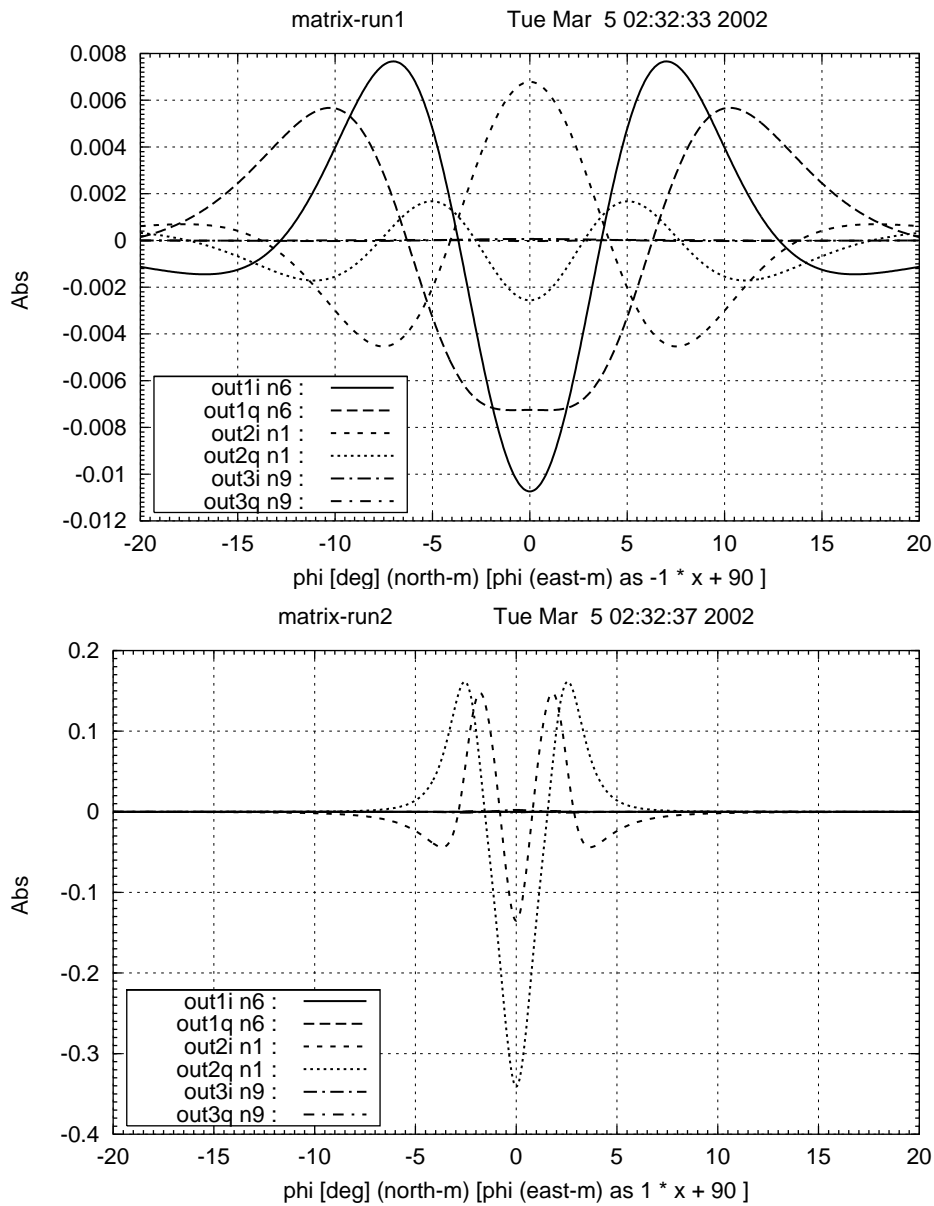
‘mkat’ then calls ‘kat’ for all these input files sequentially. If ‘mkat’ finds more than one tag name in the input file it first separates it referring to the first tag (like above) then separates each new input file depending on the second tag, etc.

¹ Originally Gerhard Heinzl wrote this script for LISO and later adapted it for FINESSE.

Example: matrix.kat

One task which can be tackled with FINESSE is to calculate error signals for the various degrees of freedom of an interferometer. **It is common to calculate a matrix of error signals.** Usually one coordinate represents the output ports (i.e. photodetectors) and the other represents input ports (mirrors which are ‘moved’), the slopes of the relevant error signals are then stored in this matrix.

Using ‘mkat’ you can easily do something similar with Finesse. Since ‘kat’ can calculate several outputs for one input file we can calculate one row of the matrix at once. In this example ‘mkat’ is used to calculate all the rows.



```
#-----
# matrix.kat example file for mkat 0.99
#
# NOTE: try 'mkat -max --quiet matrix.kat' to run this file!
#
# Plots the slope of various error signals for differential
# and common mode movement of the Michelson end mirrors
#
# freise@rzg.mpg.de 02.03.2005
#
# The "#" is used for comment lines.
#
# Power recycled Michelson Interferometer
#           with Schnupp Modulation :
#
#
#
#           <----->  north-m
#
#           n7
#
#           north-s
#
#
#           bs1           east-m
#
#           n4 .>'
#           .'.
# n0 |EOM| n1 | | n2 west-s n3 .'. n5 east-s n8 | |
#   '---'   | |           <.'
#           | |           '_,
#
#           n6
#
#           |
#           |
#           |
#           V
#
#           photo detectors
#-----

m north-m 1 0 0 n7 dump
m east-m .999 0.001 90 n8 n9
m west-m 0.9 0.1 0 n1 n2

s north-s 1201 n4 n7
s east-s 1199 n5 n8
s west-s 1 n2 n3

bs bs1 0.5 0.5 0 45 n3 n4 n5 n6

l i1 1 0 n0
```

```

mod eo1 1M 0.45 1 pm n0 n1

# three possible output ports
pd1 out1i 1M 0 n6      # main output inphase
pd1 out1q 1M 90 n6     # main output quadrature
pd1 out2i 1M 0 n1      # reflected port inphase
pd1 out2q 1M 90 n1     # reflected port quadrature
pd1 out3i 1M 0 n9      # light out of one arm inphase
pd1 out3q 1M 90 n9     # light out of one arm quadrature

# two runs of kat to cover both degrees of freedom
# of the michelson interferometer:

run1:xaxis north-m phi lin -20 20 1000
put east-m phi $dm

##-----
## mkat will do two simulations:
##
## first simulation : differential mode
## (tuning the end mirrors in differential mode)
run1:func dm = 90-$x1

## second simulation : common mode
## (tuning the end mirrors in common mode)
run2:func dm = 90+$x1

noplots y
##-----

# differentiation of the result, to give the slope of the zero
# crossings !!
diff north-m phi

yaxis abs

```


Appendix B

Some mathematics

This appendix gives some details about some of the formulae and algorithms used in FINESSE.

B.1 Hermite polynomials

The first few Hermite polynomials in their unnormalized form can be given as:

$$\begin{aligned} H_0(x) &= 1, & H_1(x) &= 2x, \\ H_2(x) &= 4x^2 - 2, & H_3(x) &= 8x^3 - 12x. \end{aligned} \tag{B.1}$$

Further polynomial orders can be computed recursively using the following relation:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x). \tag{B.2}$$

In FINESSE the functions up to H_{10} are hard-coded and for higher orders the recursion relation is used.

B.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]:

$$[\nabla^2 + k^2] E(x, y, z) = 0. \tag{B.3}$$

Two well-known exact solutions for this equation are the plane wave:

$$E(x, y, z) = E_0 \exp(-i k z), \tag{B.4}$$

and the spherical wave:

$$E(x, y, z) = E_0 \frac{\exp(-i k r)}{r} \quad \text{with} \quad r = \sqrt{x^2 + y^2 + z^2}. \tag{B.5}$$

Both solutions yield the same phase dependence along an axis (here, for example, the z -axis) of $\exp(-i k z)$. 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 :

$$E(x, y, z) = u(x, y, z) \exp(-i k z). \quad (\text{B.6})$$

Substituting this into Equation B.3 yields:

$$(\delta_x^2 + \delta_y^2 + \delta_z^2) u(x, y, z) - 2i k \delta_z u(x, y, z) = 0. \quad (\text{B.7})$$

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:

$$|\delta_z^2 u(x, y, z)| \ll |2k \delta_z u(x, y, z)|, |\delta_x^2 u(x, y, z)|, |\delta_y^2 u(x, y, z)|. \quad (\text{B.8})$$

With this approximation, Equation B.7 can be simplified to the *paraxial wave equation*:

$$(\delta_x^2 + \delta_y^2) u(x, y, z) - 2i k \delta_z u(x, y, z) = 0. \quad (\text{B.9})$$

Appendix C

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.

Most of the syntax has been the same for all versions of FINESSE. However, version 0.99 has introduced a major syntax change with respect to many previous versions: namely, the command `xparam` has been replaced by a new set of more flexible commands `set`, `func`, `put`. The manual for FINESSE 0.98 is recommended for information about the old syntax; this manual refers to the new syntax only.

C.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.

C.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 ($0 < \text{Loss} < 1$)
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 ($0 < \text{Loss} < 1$)
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 ($0 < \text{Loss} < 1$)

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 ($0 < \text{Loss} < 1$)

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

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.

- **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
```

- **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 independently**. 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).

- **pdS** : shot noise limited sensitivity

Usage is the same as for **pd**. It calculates the shot noise in the output using the DC photocurrent 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 3.5.

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 $\text{m}/\sqrt{\text{Hz}}$ 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 3.5.

- **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 4.7.1.

- **shot** : shot noise

usage : **shot name node[*]**

It calculates the shot noise in the output using the DC light power *P* as

$$\Delta P = \sqrt{\frac{2hc}{\lambda_0}} P. \quad (\text{C.1})$$

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 3.5.

- **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]`

`component` = one of the previously defined component names

`type` = type of modulation

`f` = signal frequency

`phase` = signal phase

`amp` = signal amplitude

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, angle*
- beam splitter: phase*, amplitude, angle
- space: phase*
- input: frequency*
- modulator: 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`.

C.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`

input = name of an input component

n, m = TEM_{nm} 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 TEM₀₀ 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 TEM₀₁ mode to the TEM₀₀ mode. Both fields have the same amplitude.

In order to create a pure higher order mode the TEM₀₀ 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 TEM₀₁ 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 TEM₀₀ mode should be in TEM₂₁, but with a phase offset of 90 degrees.

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 degrees
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:

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

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)

Please note that the direction parameter (x/y) only applies to the parameters related to beam size 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 TEM₀₀ and TEM₀₁, 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 TEM₀₁ and TEM₀₂ is scaled with factor 1.0
2. '0 0 * 0 1.0', '*' means 'any': the beats of TEM₀₀ with TEM₀₀, TEM₁₀, TEM₂₀, TEM₃₀,... 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.

- **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 4.7.3. 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 detector.

Without this command all photodetectors (for which 'pdtype' is not used) detect the power of all TEM modes, for example :

$$S(f_1, f_2) = \sum_{nm} 2 \operatorname{Re} (a_{nm}(f_1) a_{nm}(f_2)) = \sum_{nm} S_{nm}, \quad (\text{C.2})$$

where $a_{nm}(f)$ is the amplitude of the TEM_{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 TEM_{nm} will scale the detected power by the given factor:

$$S_{nm}(f_1, f_2) = \text{factor} \cdot 2 \operatorname{Re} (a_{nm}(f_1) a_{nm}(f_2)). \quad (\text{C.3})$$

-
- **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 vertical plane
Rcy	=	radius of curvature in horizontal plane
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 component

```
usage : map component [angle] [mapname] filename
```

component	=	mirror to which the map should be applied
angle	=	angle in degrees by which the map should be rotated
mapname	=	name to overwrite mapname found in the map file
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 4.6.2.

-
- **savemap** : saves a mirror map and the corresponding coupling coefficients to a file

```
usage : savemap component mapname filename
```

component	=	mirror to which the map has been applied
mapname	=	name of the map to be saved
filename	=	name of the file to write the data into

This command saves the computed coupling coefficients into a file. Actually, also the original data grid is saved, so that the new file contains all available information. The file format is exactly the same as required by the `map` command, see 4.6.2. Therefore `savemap` can be safely used to overwrite an existing map file (however,

old coefficients data in the existing file would be lost).

A typical combination of the `map` and the `savemap` commands might look like this:

```
% load mirror map and apply to mirror m1
map m1 45 test mymap01.txt

% overwrite old map file with new one
% thus the second time this file is run the
% coefficients do not need to be created again
savemap m1 test mymap01.txt
```

- **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]`)

`w0` = beam waist size in metres

`z` = distance to waist in metres

`q` = complex beam parameter (given as ' $z + iz_R$ ', 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:

$$q = z + iz_R,$$

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.

<i>n</i>	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]** : 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 bahavoir: **retrace** will force a retracing and **retrace off**

will prevent it, regardless of the `xaxis` settings.

Please note that the re-tracing cannot avoid all unwanted mode-mismatches.

- **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 TEM_{nm} modes.

maximum number : 1

This defines the maximum order for TEM_{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_{mnmn} 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 4.9.2 for more details.

- **kmn** : switches between different modes for computing the coupling coefficients
usage : kmn number

With ‘kmn’ the user can specify whether the coupling coefficients for TEM modes are computed with the (default) Bayer-Helms formula or by solving the convolution integral numerically:

- 0 = algorithm: Bayer-Helms [Bayer-Helms]
- 1 = verbose mode on (independent of the algorithm)
- 2 = algorithm: numeric integration if both x and y misalignments are set
- 4 = algorithm: numeric integration if x or y misalignments are set
- 8 = algorithm: numeric integration

The numeric integration uses a fast self-adapting routine [DCUHRE] 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)

C.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 maxmaximum 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 $\text{min} = -\text{max}$.

-
- **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 $W/m/\sqrt{Hz}$ (m/\sqrt{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:

$\text{diff} = (f(x+h/2) - f(x-h/2))/h$

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'.

- **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 executed 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.)

This new feature uses the mathematical expression parser Formalc 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 Formalc 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.

- **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

$$|(\$variable)| < accuracy$$

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 performs 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 parameters 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 switched 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 y_1, y_2, \dots as successive error signal values): the oscillation condition is defined as:
if $\text{abs}((y_1 + y_3 - 2*y_2)/\text{accuracy}/y_3) > \text{lockthresholdhigh}$, 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 de-

defined as:

if $\text{abs}((y1+y3-2*y2)/\text{accuracy}/y3) < \text{lockthresholdlow}$, 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 [A.1.7](#).

- **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).

- **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.

C.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 Heinzl has been the major force behind this. 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 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 and Jerome Degallaix has provided some nice test scripts for the mirror surface maps.

Paul Cochrane, who worked on the quantum noise extension for FINESSE, 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.

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 Jenmrich, James Mason, Jerome Degallaix and Daniel Shaddock. Paul Cochrane, and Roland Schilling have kindly proof-read some version of the manual (before I added all the new mistakes you probably just found).

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

- [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. [77](#), [79](#), [93](#), [157](#), [159](#)
- [Drever83] R.W.P. Drever, J.L. Hall, F.V. Kowalski, J. Hough, G.M. Ford, A.J. Munley, H.Ward: ‘Laser Phase and Frequency Stabilization Using an Optical Resonator’, *Appl. Phys.* **B 31** (1983) 97–105.
- [FINESSE] A. Freise: ‘FINESSE, Frequency domain interferometer simulation software (1999-2010), <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. [2](#)
- [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** (2003)1067–1074. [2](#)
- [Freise04] A. Freise, M. Loupas : ‘The VIRGO north arm cavity: Examples for the use of the interferometer simulation FINESSE’, available as VIRGO internal note VIR-NOT-EGO-1390-269 and on the FINESSE webpage (2004). [119](#)
- [GEO] K. Danzmann et al.: in *First Edoardo Amaldi Conference on Gravitational Wave Experiments*, Frascati (1994), (World Scientific, Singapore, 1995) 100–111. [2](#)
- [Gnuplot] T. Williams, C. Kelley et.al.: Gnuplot (1999), <http://www.gnuplot.info>. [1](#), [3](#)
- [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** (2003)473–480. [2](#)
- [Heinzel] G. Heinzel: ‘Advanced optical techniques for laser-interferometric gravitational-wave detectors’, Ph.D. Thesis, University of Hannover (1999). [29](#), [30](#), [34](#)
- [Kawabe] K. Kawabe: ‘An informal note on FINESSE: “Sidebands of Sidebands”’, internal note, available on the FINESSE webpage (2002). [29](#)
- [LISO] G. Heinzel : ‘LISO, Program for Linear Simulation and Optimization of analog electronic circuits’, MPQ Garching (1998). [2](#)
- [LUXOR] J. Harms : ‘LUXOR, graphical user interface for FINESSE’, <http://www.aei.mpg.de/~jah>. [3](#), [4](#)

- [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** (2003)985–989. [2](#)
- [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** (2003)991–998. [2](#)
- [Meers] Meers, B. J. and Strain, K. A.: ‘Modulation, signal, and quantum noise in interferometers’, *Phys. Rev. A* **44** (1991) 4693–4703. [61](#)
- [Mizuno] J. Mizuno: ‘Comparison of optical configurations for laser-interferometric gravitational-wave-detectors’, Ph.D. Thesis, University of Hannover (1995). [53](#)
- [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** (1991) 5022–5029. [61](#)
- [Num. Recipes] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: ‘Numerical Recipes in C, The Art of Scientific Computing’, Cambridge University Press, 2nd ed. (1992).
- [OPTOCAD] R. Schilling: ‘OPTOCAD, A Fortran 95 module for tracing Gaussian beams through an optical set-up, Version 0.74’, internal note (2002).
- [Rakhmanov] M. Rakhmanov, F. Bondu, O. Debieu and R. L. Savage Jr: ‘Characterization of the LIGO 4km Fabry-Perot cavities via their high-frequency dynamic responses to length and laser frequency variations’, *Class. Quantum Grav.* **21** (2003)487–492. [2](#)
- [Rüdiger] A. Rüdiger: ‘Phasenbeziehungen an einem symmetrischen Strahlteiler’, internal note (1978). [29](#)
- [Rüdiger02] A. Rüdiger: ‘Shot Noise in Interferometers used for gravitational wave measurements’, unpublished.
- [Siegman] A.E. Siegman: ‘Lasers’, University Science Books, Mill Valley (1986), see also the Errata at www-ee.stanford.edu/siegman/lasers_book_errata.pdf. [69](#), [71](#), [94](#), [143](#)
- [STAIC] Software Tools for Advanced Interferometer Configurations, <http://www.phys.ufl.edu/LIGO/LIGO/STAIC.html>. [2](#)
- [Sparse] K.S. Kundert, A. Sangiovanni-Vincentelli: ‘Sparse, A Sparse Linear Equation Solver’, University of California, Berkeley (1988). [3](#), [116](#)
- [KLU] Davis T. A: ‘CSparse, CXSparse, KLU, and BTF: Direct Methods for Sparse Linear Systems’, SIAM, Philadelphia (2006). [116](#)
- [VIRGO] C. Bradaschia et al.: Nuclear Instruments and Methods in Physics Research A **289** (1990) 518–525.

- [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> 77, 102
- [Willke01] B. Willke et al.: ‘The GEO 600 gravitational wave detector’, *Class. Quantum Grav.* **19** (2002)1377–1387. 2
- [DCUHRE] J. Berntsen, T. O. Espelid and A. Genz: ‘Algorithm 698; DCUHRE: an adaptive multidimensional integration routine for a vector of integrals’, *ACM Transactions on Mathematical Software (TOMS)*, **17**, 4 (1991) 452–456. 159
- [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. 43, 46