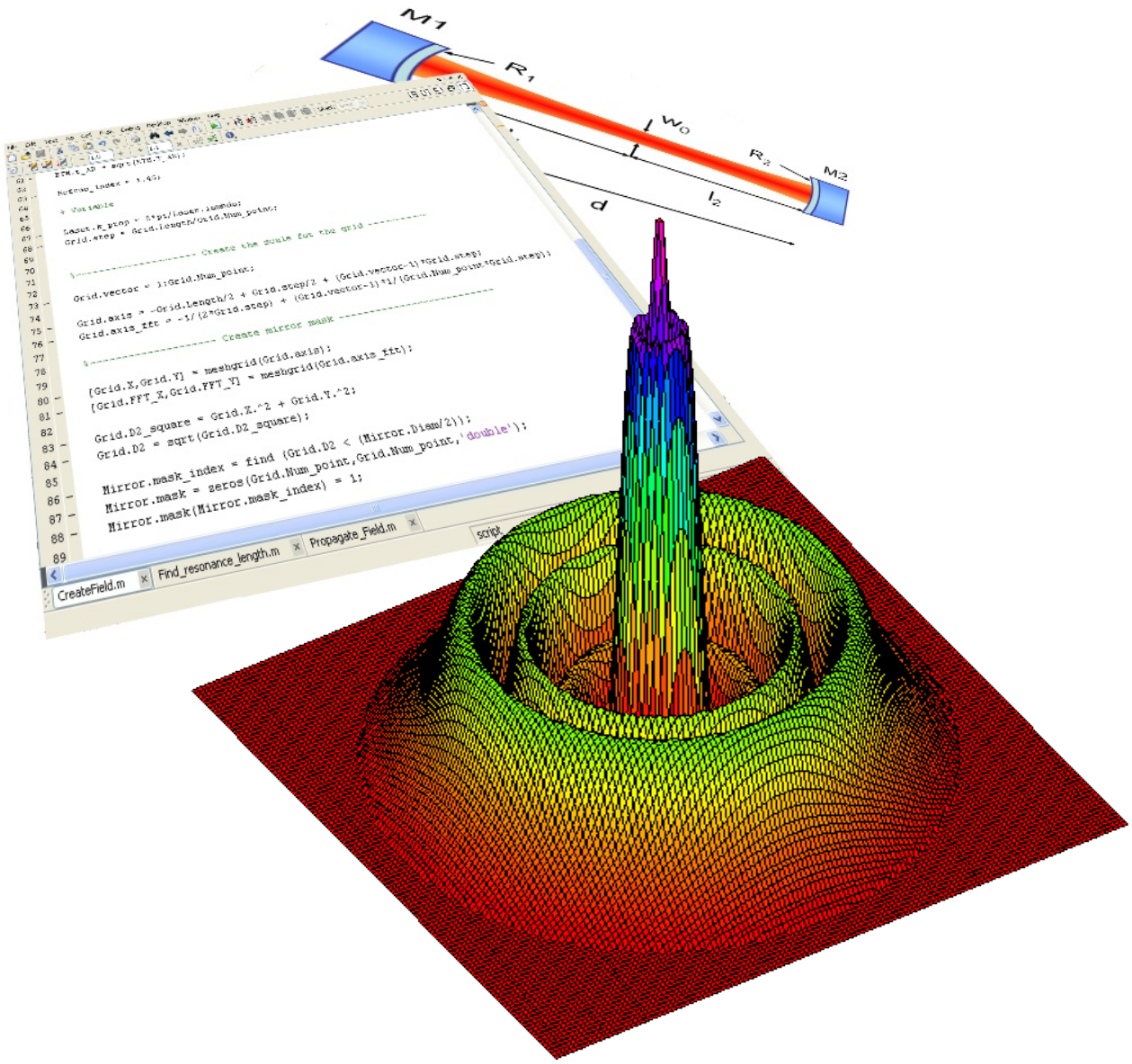
OSCAR

A Matlab-based optical FFT code



Jerome Degallaix

OSCAR

The Non-Definitive Guide

First edition	July 2008
Second edition	October 2008
Third edition	April 2009
Fourth edition	January 2010
Fifth edition	January 2012
Sixth edition	June 2012
Seventh edition	January 2013
Eight edition	September 2013
Ninth edition	December 2013
Tenth edition	July 2014
Eleventh edition	December 2014
Twelfth edition	September 2016
Thirteenth edition	August 2019
Fourteenth edition	June 2020
Fifteenth edition	March 2022

OSCAR, the Matlab code, this documentation as well as the examples were written by:
Jerome Degallaix
Laboratoire des Matériaux Avancés (LMA)
Université Claude Bernard Lyon I
7, Avenue Pierre de Coubertin
69622 Villeurbanne
France

OSCAR is an optical FFT code used to calculate the steady state optical field circulating in Fabry-Perot cavities. The code can integrate non-spherical mirrors and any arbitrary input fields. Typical applications for OSCAR have been: calculation of thermal lensing effect and calculation of diffraction loss and cavity eigenmodes for arbitrary beam shapes. One great advantage of OSCAR is the simplicity and flexibility of the code, everyone with only minimal knowledge of Matlab can easily modify OSCAR code to suit specific purposes.

If you find a bug when using OSCAR, please feel free to contact the author. Do not hesitate also to report any unclear part from this manual or from the code itself. Of course, I am also open to suggestions for any possible improvement.

Contents

Chapter 1

Principle of the FFT code

In this chapter, we detail the two main steps used to propagate an electric field in a Fabry-Perot cavity. Firstly we will understand how the propagation of an arbitrary electric field can be simulated using a Fourier transform. And secondly, we will see how any wavefront distortion encountered by the electric field can be included in the numerical simulations. Whenever possible, some simple Matlab codes are presented to show the numerical implementation of the algorithm. For clarity, the snippets of code are **not** optimized.

1.1 Why OSCAR ?

OSCAR is a FFT code which is able to simulate Fabry-Perot cavities with arbitrary mirror profiles. One of the key features of OSCAR is the possibility to easily modify the code to suit the user purpose. For this reason, OSCAR is written with the Matlab language, one can import/export files (mirror maps or cavity eigenmode profiles for example), create a ring cavity, create batch file or plot 2D optical fields with little programming skill[1]. The fact that I encourage everyone to understand the code of OSCAR in details and to also modify it, is the main reason for this lengthy manual.

1.1.1 Origin of OSCAR

I started writing and using OSCAR during my PhD where I had to simulate the effects of thermal lensing in high optical power cavities. Wavefront

[1]The code can also be used with the free Matlab-alternatives such as Sci-Lab or Octave with only minor modifications. Even a minimal Python version exists.

distortions induced by the optical absorption have seldom pure spherical profiles, which means it is delicate to quantify thermal lensing effects with modal expansion code (but still possible). To simulate the effects of the optical absorption on the mirror, I was using the FEM package Ansys which can calculate the temperature distribution inside the mirror substrate or the surface deformations due to the absorption of a Gaussian beam. From the Ansys results, it is straightforward to calculate the wavefront distortion induced by thermal lensing. However, at this stage, I was faced with a problem : how can I estimate the effects of the thermal lens if it has to be inserted in a Fabry-Perot cavity ? How much will the circulating power decrease ? Will the cavity optical modes keep their Gaussian profiles ? I needed an optical simulation tool which can be used with any arbitrary mirror profile. OSCAR was born. For the inquisitive reader, the name OSCAR is the acronym of Optical Simulation Containing Ansys Results.

The first version of OSCAR was written with the software IGOR. This code was then translated in Matlab and used to calculate diffraction losses by my UWA colleague Pablo Barriga. Finally, I re-wrote and optimized the Matlab code to substantially decrease the OSCAR computational time. During the year 2011, OSCAR was totally revamped using the Oriented Object capability of Matlab giving rise to the Version 3.0.

1.1.2 Which results could I get from OSCAR ?

OSCAR is a versatile tool to simulate Fabry-Perot cavities. The following is a (non-exhaustive) list of the results which can be obtained with OSCAR:

- calculate the Gouy phase shift between higher order optical modes. It may be useful for flat beams for example, where no analytical calculations of the Gouy phase shift has been derived yet (as far as I know)
- calculate the coupling loss between the input beam and the cavity eigenmodes in case of mode mismatching
- calculate the circulating beam (intensity and profile) inside the cavity, which may be different from the eigenmodes if the cavity has a low finesse
- calculate diffraction loss and eigenmodes of cavity for arbitrary mirror profiles
- calculate the effects of imperfect optics, due to the micro-roughness of the coating surface or thermal lens inside the substrate of the mirrors.

Even more complicated configuration could be simulated such as dual recycled Michelson with Fabry-Perot arm cavities (topology of advanced laser gravitational wave detectors). Although the code is more complex and lacking exhaustive documentation, it is available on demand.

1.1.3 What OSCAR is not for

OSCAR is designed to simulate anything that can be derived from the steady state, classical, optical field circulating inside a Fabry-Perot cavity. It means OSCAR does not take into account radiation pressure or quantum effects (no shot noise calculations).

1.2 Propagation of an optical field

In this section we will see how it is possible to propagate a coherent electric field, i.e. a laser beam, with the help of the Fourier transform. The principle of the FFT optical propagation code is similar to the Fourier transformation method used to calculate the response of a linear system to an arbitrary function $f_i(t)$. The function $f_i(t)$ is expressed as a continuous superposition of harmonic functions of different frequencies ν:

$$f_i(t) = \int_{-\infty}^{\infty} \widetilde{f}_i(\nu) \exp(2\pi j\nu t) d\nu \tag{1.1}$$

Where $\widetilde{f}_i$ is the Fourier transform of the function f_i. The function $t \to \exp(2\pi j\nu t)$ is the harmonic function of frequency ν. These are the basic functions used to expand f_i. If the response of the system is known for each elementary harmonic function $\exp(2\pi j\nu t)$, the response of the system to the input f_i can be derived using three steps. First step, the function f_i is expanded in the sum of basic harmonic functions by doing a Fourier transformation. Second step the response of the system to each harmonic function is calculated. This is done simply by a multiplication in the frequency domain. Finally, third step, the output of the system to the input f_i is derived by doing the inverse Fourier transformation of the output harmonic functions (step similar to the equation 1.1).

The FFT optical propagation code follows exactly the principle described in the previous paragraph. In this case the elementary functions are plane waves of different spatial frequencies.

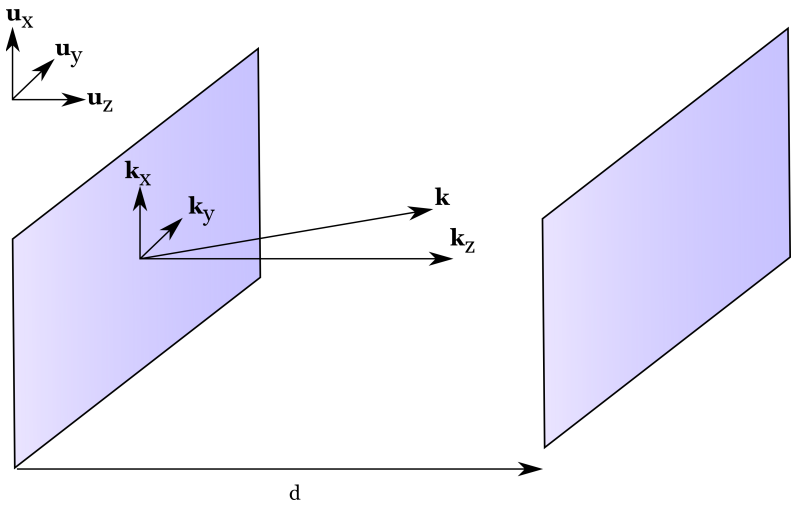


Figure 1.1: Propagation of a plane wave along the z axis.

1.2.1 Propagation of a plane wave

We can first understand how to propagate of an elementary plane wave u through free space. At $z = 0$, u can be simply written:

$$u(x, y, 0) = \exp(-jk_x x - jk_y y) \tag{1.2}$$

with $k_{x/y}$ the propagation constant along the x/y axis. Equation (1.2) can also be written:

$$u(x, y, 0) = \exp(-j2\pi(\nu_x x + \nu_y y)) \text{ with } \nu_x = k_x/(2\pi) \text{ and } \nu_y = k_y/(2\pi) \tag{1.3}$$

If u is written as shown in equation (1.3), it is easy to recognize u as a harmonic function of two variables (x, y) with respective spatial frequencies [2] (ν_x, ν_y). If we propagate u along the z axis over a distance d:

$$u(x, y, d) = \exp(-jk_x x - jk_y y - jk_z d) \tag{1.4}$$

with $k = (k_x^2 + k_y^2 + k_z^2)^{\frac{1}{2}} = 2\pi/\lambda$. If we consider a wave traveling in a direction close to the z axis (paraxial approximation) as shown in figure 1.1, $k_z \gg k_x$

[2] The term spatial frequency ν must be understood as the number of wavelengths per unit of length $\nu = 1/\lambda = k/2\pi$

and $k_z \gg k_y$, then k_z can be written as:

$$k_z = (k^2 - k_x^2 - k_y^2)^{\frac{1}{2}} \simeq k - \frac{(k_x^2 + k_y^2)}{2k} \quad (1.5)$$

$$\simeq k - \lambda\pi(\nu_x^2 + \nu_y^2) \quad (1.6)$$

So we can rewrite equation (1.4) as:

$$u(x, y, d) = u(x, y, 0) \exp(-j(k - \lambda\pi(\nu_x^2 + \nu_y^2))d) \quad (1.7)$$

The equation (1.7) is the foundation of the FFT propagation code. It tells us that the propagation of a plane wave along the z axis over a distance d can simply be represented by a phase shift. The function $d \rightarrow \exp(-j(k - \lambda\pi(\nu_x^2 + \nu_y^2))d)$ could also be seen as the propagation operator for an input plane wave traveling a distance d under the paraxial approximation. The term $\exp(-j(kd))$ represents the phase shift of a plane wave propagating along the z axis and the term $\exp(j\lambda\pi(\nu_x^2 + \nu_y^2)d)$ adds a phase correction to take into account the fact that the wave propagates with a small angle with respect to the z axis[3].

We noticed in equation (1.3), that the elementary plane wave $u(x, y, 0)$ can also be seen as a harmonic function with spatial frequency ν_x and ν_y. To keep the analogy developed in the introduction of this chapter with the classical time-domain Fourier transform, the plane wave is equivalent to the elementary harmonic function $t \rightarrow \exp(2\pi j\nu t)$ of frequency ν. Since we know how to propagate a plane wave, we understand now how it may be possible to propagate any arbitrary field if we can manage to expand it as a superposition of elementary plane waves.

1.2.2 Propagation of an arbitrary field

Equation (1.7) tells us how to propagate a plane wave. So to propagate an arbitrary electric field E, we need to know how to expand the electric field onto the set of plane waves $\exp(-j2\pi(\nu_x x + \nu_y y))$. We would like to find $\widetilde{E}(\nu_x, \nu_y)$ such that :

$$E(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \widetilde{E}(\nu_x, \nu_y) \exp(-j2\pi(\nu_x x + \nu_y y)) d\nu_x d\nu_y \quad (1.8)$$

[3]The angle θ_x between the direction of propagation and the x axis is $\theta_x = sin^{-1}(k_x/k)$. In the case of small angles, it is simply $\theta_x = \lambda\nu_x$

With $\widetilde{E}(\nu_x, \nu_y)$ the complex amplitude of the component of the field E with spatial frequency (ν_x, ν_y). We recognize equation (1.8) as an inverse 2D Fourier transform[4]. Using the properties of the Fourier transform[2] we deduce the expression for $\widetilde{E}(\nu_x, \nu_y)$:

$$\widetilde{E}(\nu_x, \nu_y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} E(x, y)\exp(j2\pi(\nu_x x + \nu_y y))dxdy \tag{1.9}$$

Combining equations (1.7), (1.8) and (1.9) , we know how to propagate in free space a transverse electric field E from $z = 0$ to $z = d$. To calculate the resulting field after the propagation, three steps are required:

1. Decomposition of the field $E(x, y, 0)$ into a sum of elementary plane waves. Mathematically, this step represents a 2D Fourier transformation.

$$\widetilde{E}(\nu_x, \nu_y, 0) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} E(x, y, 0)\exp(j2\pi(\nu_x x + \nu_y y))dxdy \tag{1.10}$$

2. Propagation of each plane wave, which is equivalent to adding a phase shift in the frequency domain.

$$\widetilde{E}(\nu_x, \nu_y, d) = \widetilde{E}(\nu_x, \nu_y, 0)\exp(-j(k - \lambda\pi(\nu_x^2 + \nu_y^2))d) \tag{1.11}$$

3. Re-composition of the electric field from the propagated plane waves. This step is in fact an inverse Fourier transformation.

$$E(x, y, d) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \widetilde{E}(\nu_x, \nu_y, d)\exp(-j2\pi(\nu_x x + \nu_y y))d\nu_x d\nu_y \tag{1.12}$$

1.2.3 My first Matlab FFT code

There is an essential point to realize before implementing the three analytical steps described in the previous section. The computer does not deal with continuous electric fields, which means that all the data fields have to be discretized. For example the amplitude of a Gaussian beam will be represented

[4] In fact, if we respect the convention found in signal processing technique, equation (1.8) is not an inverse Fourier transform but a Fourier transform[1]. It is not tragic, since we will stay consistent with the convention presented here.

Listing 1.1: Discretization of a Gaussian beam

```
Grid.Num_point = 128;                                % Number of points in one side the grid
Grid.Length = 0.10;                                  % Physical dimension of the grid in meters
Grid.step = Grid.Length/Grid.Num_point;              % Physical size of one pixel of the grid

Grid.vector = 1:Grid.Num_point;                      % Grid.vector = 1 2 3 ... Grid.Num_point
% Calculate the spatial scale used for each pixel:
Grid.axis = -Grid.Length/2 + Grid.step/2 + (Grid.vector-1)*Grid.step;

Field.Gaussian = zeros(Grid.Num_point,Grid.Num_point,'double');
Laser.amplitude = 1;                                 % Arbitrary amplitude
Laser.waist = 0.01;                                  % waist of the laser beam in meters

% Fill the matrix representing the (real) Gaussian beam:
for m = 1:Grid.Num_point
    for n = 1:Grid.Num_point
        Field.Gaussian(m,n) = Laser.amplitude * exp(-(Grid.axis(m)^2+Grid.axis(n)^2)/Laser.waist^2);
    end
end
```

by a square matrix (called also grid later), each point of the matrix (called also pixel for convenience) will represent the amplitude of the Gaussian beam at a defined location. A 2D plot of such a matrix is shown in the top left corner of the figure 1.2.

Practically, the discretization process is governed by the choice of 2 parameters: the physical size represented by the matrix and the number of points in the matrix. For example, to discretize a laser beam with a beam radius of 1 cm we can use a matrix of size 128×128 representing an area of 10 cm by 10 cm. This example can be implemented in Matlab in a straightforward way as shown in the listing 1.1.

If the scale of the matrix (called *Grid.axis* in the listing 1.1) representing the amplitude distribution is easy to understand, a more delicate point is the scaling of the Fourier transform of the input beam. This scaling of the spatial frequency is required since we need to know the spatial frequency represented by each pixel of the discrete Fourier transform of the Gaussian beam. Concretely we need to know the discrete values of ν_x, ν_y from equation 1.10.

First thing to understand is that the discrete Fourier transform of a 2D complex matrix is also a 2D complex matrix with the same dimensions [3]. The low spatial frequencies are located in the middle of the matrix and the high spatial frequencies on the edge. Typically if the original matrix has for

dimensions $N \times N$, the spatial frequency 0 (the average component) is located at the index $(N/2+1, N/2+1)$[5]. Meanwhile, the frequency separation $\Delta\nu$ between 2 adjacent pixels of the Fourier matrix is:

$$\Delta\nu = \frac{1}{Grid.Length} = \frac{1}{N \times Grid.step} \tag{1.13}$$

With *Grid.Length* and *Grid.step* the variables defined in the listing 1.1. So the minimal (negative) spatial frequency calculated is $-N/2 * \Delta\nu$ often called the Nyquist frequency and the maximal spatial frequency is $(N/2-1) * \Delta\nu$. An example for the frequency scale of the Fourier transform of the matrix is presented in the top right plot of the figure 1.2.

Special attention must be taken to understand the vertical and horizontal scales in the figure 1.2. The numbers written for the scales are in fact the value of the scale between 2 pixels as can be seen by zooming on the ticks. For example, to know what is the spatial frequency of the first top row (horizontal line) of the Fourier matrix on the top right plot which represents the maximal spatial frequency, we have to take the average of the top two ticks. So the maximal spatial frequency is $1/2*(96.875+90.625) = 93.75\text{m}^{-1}$, which is as expected equal to $(N/2-1) * \Delta\nu = 15/0.16 = 93.75$

One natural question we could ask is : what is the good size of the grid ? The size of the grid must be large enough to represent faithfully the Gaussian beam, no substantial energy from the beam must lay outside the grid. So the dimension of the grid must be *at least* 3-4 times bigger than the biggest beam diameter encountered. A large grid size is also important to sample properly the low special frequency as shown at the bottom of figure 1.2. Remarkably, even if the laser beam radius increases as the beam propagates from its waist, the amplitude of the Fourier transform is constant along the propagation. Indeed the propagation as shown in equation 1.11 is simply represented by a phase shift in the Fourier domain.

After the physical size of the grid has been chosen, the second important parameter to be decided is the number of points in the grid. Because of the way the discrete Fourier transform is calculated in the FFT algorithm, it is strongly recommended to choose the number of points for the side of the matrix to be a power of 2. Usually a good compromise between speed and accuracy is given for N =64,128 or 256. The influence of the number of points

[5]In fact the FFT algorithm returns the Fourier transform of the input matrix with the low spatial frequency spread at the four corners of the matrix. However for a better readability, the low frequencies are then shifted back to the center of the matrix.

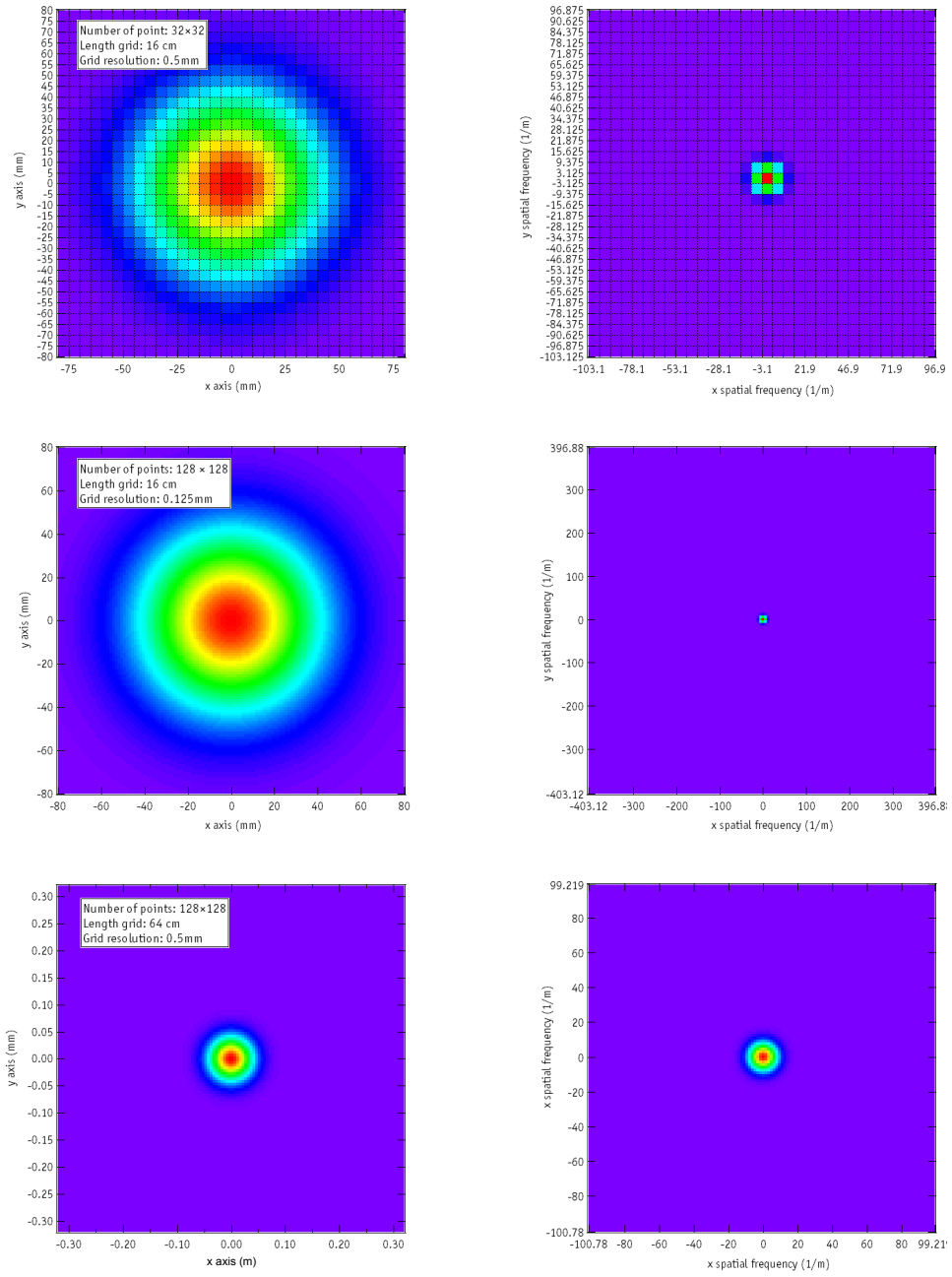


Figure 1.2: Example of the influence of the physical size of the grid and the number of points used for the representation of a Gaussian beam (left column) and its Fourier transform (right column). As expected intuitively, to have a proper representation of both the Gaussian beam and its Fourier transform, it is essential to have simultaneously a large space window (parameter: *Grid.Length*) and a large number of points (parameter: *Grid.Num_point*). For this example the waist of the laser beam is 4 cm.

on the accuracy of the results can (and should) always be checked by doing the same simulations with different meshings of the grid.

The size of the grid divided by the number of pixels is the dimension represented by one pixel, in other word the resolution of the grid (variable *Grid.step*). Of course, the resolution of the grid must match the size of the physical feature we would like to simulate. For example it is useless to try to simulate the effect of features having a size 1 mm in a mirror map with a grid resolution of 1 cm.

We just saw how a square matrix can be used to represent a discrete electric field, so now we can try to simulate its propagation using the 3 consecutive steps described by equations (1.10), (1.11) and (1.12). During the second step, the Fourier transform of the electric field is multiplied by a complex number depending of the distance of propagation and also the spatial frequency. Practically, this multiplication is achieved by multiplying pixel by pixel 2 matrices : the Fourier transform matrix and a propagation matrix. The propagation matrix is usually defined beforehand and only once, since it will be used repeatedly as we will see later in section 1.4. The Matlab code at the core of the FFT code is presented in the listing 1.2, it is the direct sequel from the previous listing where we defined the matrix used to represent the electric field.

The propagation of an electric field using a FFT code is an extremely powerful tool. With the FFT code we can propagate any arbitrary profile of the laser beam, not only the beams from the usual Hermite-Gaussian set. Such an example is presented in figure 1.3. The code used to produce these two plots is given with the OSCAR distribution, the name of the Matlab script is My_First_FFT_code.m. The initial field is a theoretical square of uniform amplitude (left plot). The resulting field after the propagation in free space over 100 m is shown in the right plot. A similar result could have been obtained based on the propagation of Hermite-Gaussian modes. However to have an accurate representation of the initial field, a very large number of the higher order modes must be taken into account, which requires large amounts of computer processing power. The field is discretized on a 1024×1024 matrix. The physical size of the grid is 16 cm $\times$ 16 cm.

We arrive now at the end of this section, which is dedicated to understanding how we can numerically simulate the propagation of an arbitrary laser beam. However having a code only capable of propagating a beam is seldom useful. We are usually more interested in simulating real optical sys-

Listing 1.2: The code used to propagate the matrix Field.Start

```
% Distance of propagation in meters
Distance_prop = 100;

% Spatial frequency of the pixels in the Fourier space
Grid. axis_fft = -1/(2*Grid.step) + (Grid.vector-1)*1/(Grid.Num_point*Grid.step);

% Define the propagation matrix

Mat_propagation = zeros(Grid.Num_point,Grid.Num_point,'double');

for m = 1:Grid.Num_point
    for n = 1:Grid.Num_point

        Mat_propagation(m,n) = exp(i*(-Laser.k_prop*Distance_prop + ...
            pi*Laser.lambda*(Grid.axis_fft(m)^2 + Grid.axis_fft(n).^2)*Distance_prop));

    end
end

%------------------- Propagate the field ----------------------------

Field.Fourier = fftshift ( fft2 (Field.Start ));          % Do the Fourier transform of the input field
Field.Fourier = Field.Fourier .* Mat_propagation; % Do the propagation in the frequency domain
Field.End = ifft2( ifftshift (Field.Fourier ));           % Do the inverse Fourier transform

% As a result Field.End represents Field.Start propagated over 100 meters
```

tem with mirrors, apertures and imperfect optics. How to include optics in the code is the subject of the next section.

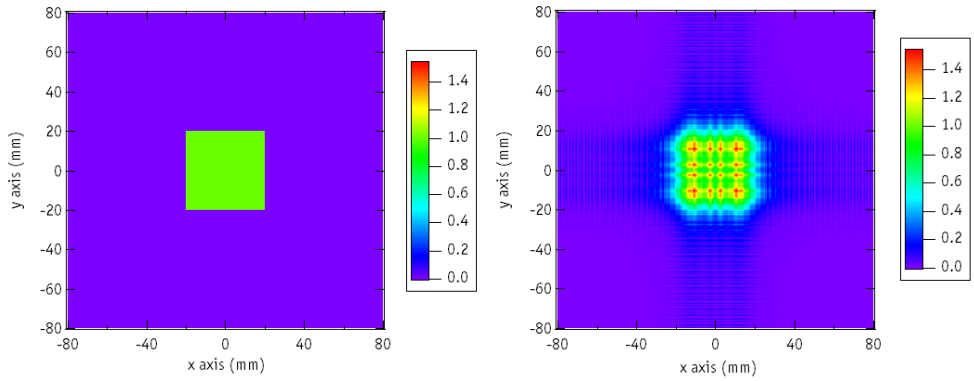


Figure 1.3: Propagation of an uniform square of light using our FFT propagation code. The initial square light field is presented on the left plot. The field resulting from the propagation of the initial field over 100 m is shown on the right plot. Structures similar to the Hermite-Gaussian modes begin to emerge when the square field propagates in free space.

1.3 Adding realistic optics

It is time now to introduce in our simulation two essential optical components: mirrors and lenses. These components alter the beam wavefront radius of curvature and are therefore used for beam shaping (in other words : they make the laser beam smaller or bigger). This can be easily implemented in OSCAR as we will discover in the following paragraphs.

1.3.1 Arbitrary wavefront distortion

Any wavefront distortion can be characterized by its induced optical path length difference $\Delta OPL(x, y)$. In a general manner, when the laser beam crosses a medium of non-uniform refractive index $n(x, y, z)$ the optical path length $OPL(x, y)$ along the optical axis parallel to the s direction can be defined as:

$$OPL(x, y) = \int_0^L n(x, y, z)dz \tag{1.14}$$

With L the length of the medium. Since we are not interested in any constant offset due to the optical path length, it is often more relevant to introduce the optical path length difference ΔOPL as:

$$\Delta OPL(x, y) = \int_0^L n(x, y, z)dz - \int_0^L n(0, 0, z)dz \tag{1.15}$$

The laser field E_i passing trough an element inducing a wavefront distortion characterized by $\Delta OPN(x, y)$ get an additional space dependent phase shift according to:

$$E_o(x, y) = E_i(x, y) \exp\left(-jk\Delta OPL(x, y)\right) \tag{1.16}$$

As we can see the effect of the wavefront distortion can be implemented in the physical space and it is not related to any Fourier transform. In fact in any optical FFT code, the Fourier transform is only used to propagate the electric field over a certain distance. All other calculations are made in the usual physical space.

1.3.2 Mirrors

One of the most useful wavefront distortions is the one induced by mirrors. A mirror is a reflective spherical surface which is used to steer (flat mirror) or focus the beam (convergent or divergent mirrors). From simple geometrical

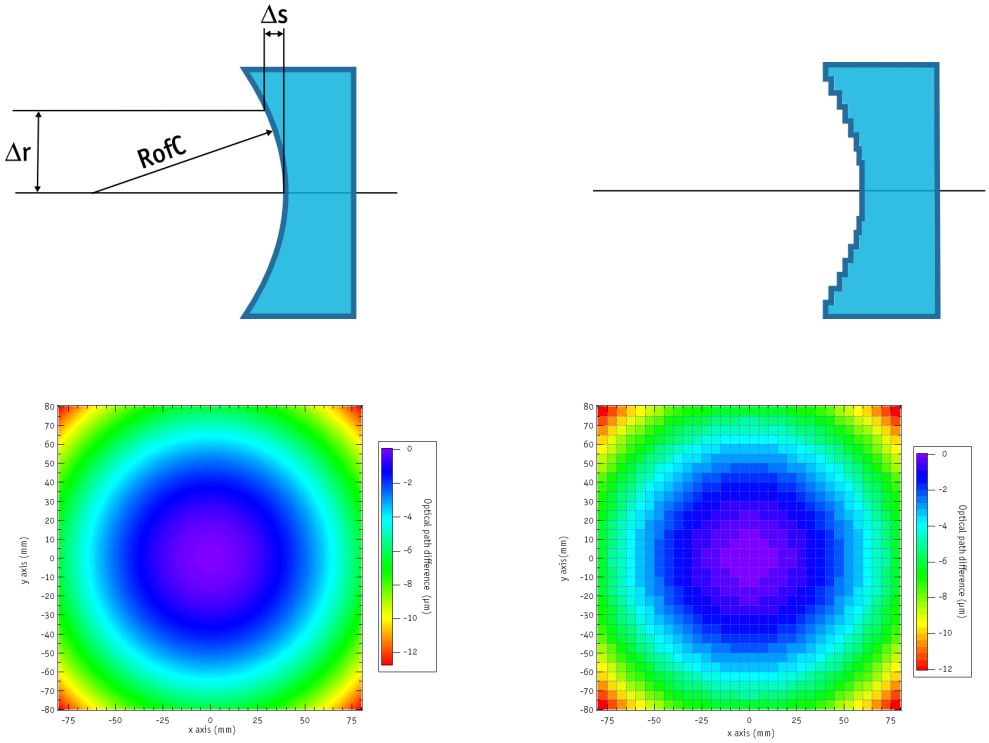


Figure 1.4: Example of the optical path difference induced by 1 km radius of curvature mirror. The optical path difference is simply twice the sagitta of the mirror. The right plot is the left plot discretised on a grid of 32×32 pixels.

considerations (see the top left plot in figure 1.4) we can calculate the change in sagitta Δs as a function of Δr the distance from the mirror center:

$$\Delta s = RofC - \sqrt{RofC^2 - \Delta r^2} \tag{1.17}$$

With $RofC$ the radius of curvature of the mirror. The optical path difference is simply twice the sagitta:

$$\Delta OPL(x, y) = 2\left(RofC - \sqrt{RofC^2 - (x^2 + y^2)}\right) \tag{1.18}$$

Of course since we are using a numerical simulation, the optical path difference representing the mirror has also to be discretized using the same grid

as the one used for the laser beam. A discretized optical path difference for a mirror is represented on the right part in the figure 1.4.

There is no difficulty in creating with Matlab the matrix used to represent the optical path difference induced by a mirror. The value of the optical path as a function of the coordinate (x, y) has previously been shown in equation 1.18. The direct Matlab implementation is presented in the listing 1.3. By definition, in OSCAR a concave mirror has a positive radius of curvature, which means the optical path difference is negative.

Listing 1.3: The code used to create the mirror matrix

```
% Definition of the mirror radius of curvature in meters
Mirror.RofC = 1000;

% Create mirror grid
Mirror.OPL = zeros(Grid.Num_point,Grid.Num_point,'double');


for m = 1:Grid.Num_point
    for n = 1:Grid.Num_point

        Radius_sqr = (Grid.axis(m)^2+Grid.axis(n)^2);
        Mirror.OPL(m,n) = -2*(Mirror.RofC - sqrt(Mirror.RofC^2 - Radius_sqr));

    end
end
```

If the mirror is not perfectly spherical because of thermal lensing effect or because the mirrors are part of a flat beam cavity, we simply have to modify the equation 1.18 accordingly by adding the known deviation.

1.3.3 Lenses

In OSCAR, we use the exact same procedure as for mirrors to simulate lenses. The optical path difference induced by the lens is the same as that induced by a mirror whose radius of curvature is twice the focal length of the lens we wish to simulate. The fact that the lens is used in transmission and a mirror in reflection is not relevant in OSCAR. Indeed the evolution of the laser beam confined between two mirrors can always be simulated by the propagation of a laser beam passing through a periodic system of lenses[4].

1.3.4 Aperture

Apertures can be represented by two complementary physical areas: one area transmits integrally the light falling on it whereas the other area blocks integrally the light. Apertures are useful to simulate correctly finite size optics. In OSCAR, the optical path difference representing a mirror is defined over the whole calculation grid independently of the real size of the mirror. To simulate a finite size mirror, we multiply the reflected field by an aperture which has the same diameter as that of the mirror. The aperture simulates the fact that any light falling outside the mirror is lost.

Practically, an aperture $A(x, y)$ is represented by matrix of 0 and 1. A 0 at the position (x, y) indicates that the light is blocked and a 1 indicates that the light is transmitted. An example of a circular aperture is presented in figure 1.5. Thus, to simulate the reflection from a finite size mirror, we can include the aperture effect in the previous equation 1.16:

$$E_o(x, y) = E_i(x, y) \exp\left(-jk\Delta OPL(x, y)\right) A(x, y) \tag{1.19}$$

As for the optical path difference induced by the mirror, we can also define a matrix representing the aperture. In the OSCAR code this matrix is called $Mirror.mask$. This matrix is only filled with 0 and 1, 1 when the pixel is inside the mirror and 0 otherwise. The simple Matlab code to create a circular aperture in OSCAR is described in the listing 1.4.

Listing 1.4: The code used to create a circular aperture

```
% Define aperture diameter in meters
Aperture_diameter = 0.1;

Mirror.mask = zeros(Grid.Num_point,Grid.Num_point,'double');

for  m = 1:Grid.Num_point
    for  n = 1:Grid.Num_point

        Radius = sqrt(Grid.axis(m)^2+Grid.axis(n)^2);
        if  (Radius < Aperture_diameter/2)
            Mirror.mask(m,n) = 1;
        end
    end
end
```

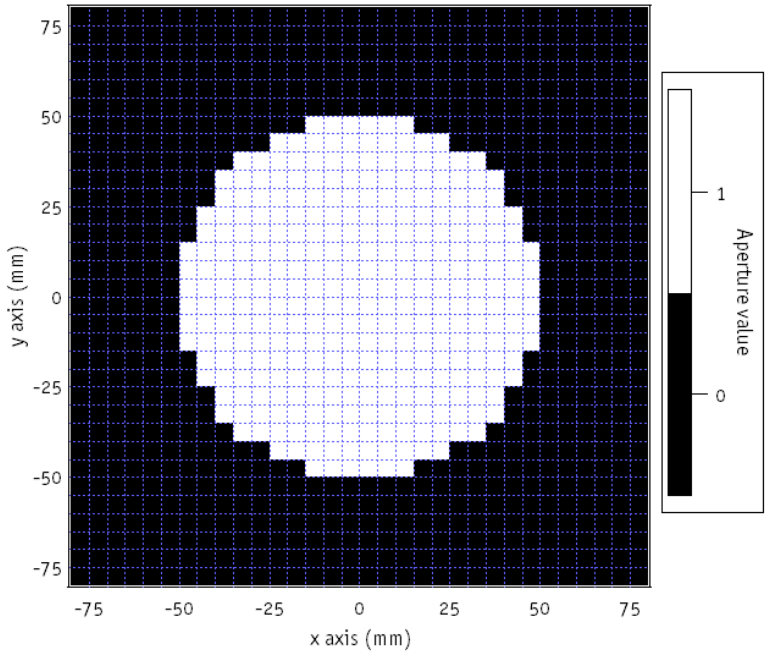


Figure 1.5: Plot of the 2D matrix representing a circular aperture of diameter 10 cm. The number of points for the grid is 32×32 points and the physical size of the grid is 16 cm $\times$ 16 cm.

1.3.5 Code implementation

To simulate the reflection of an electric field by a mirror (or a transmission through a lens), we define a function called *Propa_mirror*. The function takes as parameters the input electric field, the optical path difference induced by the mirror and the reflectivity of the mirror as shown in the listing 1.5 . The output of the function is the electric field after the reflection on the mirror. The function is a direct implementation of the equation 1.16.

The aperture of the mirror is also included in the reflection ; however it is not an argument of the function *Propa_mirror* since we suppose that the mirrors have all the same diameter, so the aperture matrix is constant[6]. In the function, the reflectivity of the mirror is scalar, which means the reflectivity is homogeneous and constant over the mirror surface. The reflectivity of the

[6]The function can easily be modified if the mirrors have different sizes.

mirror can also be defined as a matrix if the reflective coating is not perfectly uniform.

Listing 1.5: The function used to simulate the reflection of an electric field by a mirror

```
function Output = Propa_mirror(Wave_field, Wave_mirror, reflectivity)

global Mirror;
global Laser;

Output = Wave_field .* exp(i * Wave_mirror*Laser.k_prop) * reflectivity .* Mirror.mask;
```

1.4 Simulating a Fabry-Perot cavity

Since we have seen in the two last sections how to propagate a laser beam in free space (section 1.2) and how to simulate the reflection by a mirror (section 1.3), we have everything we need to simulate a Fabry-Perot cavity.

A Fabry-Perot cavity is usually constituted by two mirrors facing each other. Between these two mirrors, a light field is circulating, bouncing back and forth between the two reflective coatings. One of the main interests of the Fabry-Perot is that the optical power of the circulating field can be much higher than the power of the input field. With OSCAR, it is possible for a given input field to calculate the total circulating power, reflected power and transmitted power as well as the spatial profile of all the light fields.

The method used by OSCAR to calculate the circulating field in a Fabry-Perot cavity is well known for most readers. Indeed, the same method is often used in undergraduate lectures to calculate analytically the circulating field in the cavity[5]. OSCAR calculates the circulating field by propagating back and forth the laser beam between the two mirrors and then summing all the fields at one particular plane as shown in figure 1.6.

In more details, we can write the OSCAR algorithm used to compute the circulating power. Using the notation from the figure 1.6 the different consecutive steps can be described as:

1. Define the cavity parameters as well as the mirror profiles and the input beam.

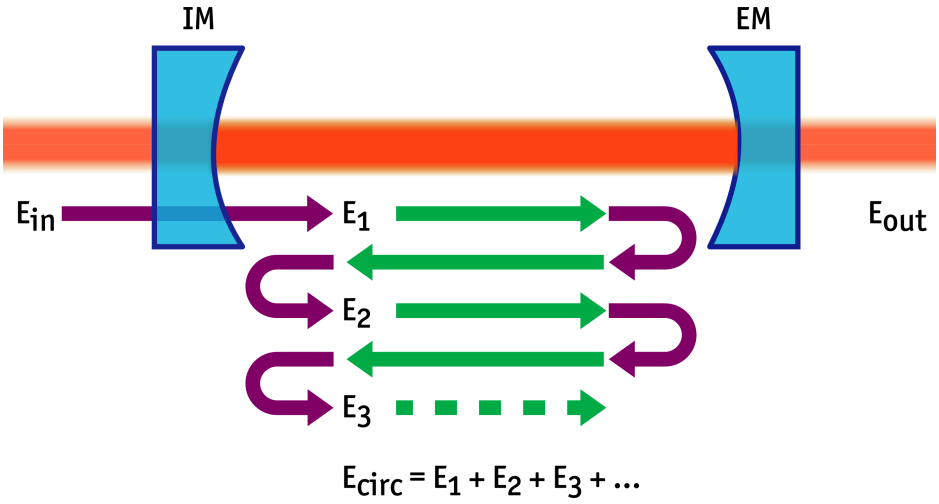


Figure 1.6: Description of the algorithm used in OSCAR to calculate the circulating field in a Fabry-Perot cavity. The violet arrows represent a phase change for the light field which is described by equation 1.16 and the green arrows represent the propagation of the light field using a FFT code. From E_i to E_{i+1}, the light field has undergone one round-trip in the cavity. IM and EM stand respectively for Input Mirror and End Mirror.

2. Propagate the input beam E_{in} through the input mirror . For this purpose the input mirror can be assimilated to a lens, so the listing 1.5 can be used. As a result, we obtain the field E_1.

3. After one round-trip in the cavity, the field E_1 becomes E_2. One round-trip in the cavity consists specifically of one propagation through the cavity length using the FFT code, one reflection on the end mirror, another propagation back to the input mirror and then finally a reflection on the input mirror.

4. Repeat the last operation to create the set of electric field E_i.

5. Then sum all the field E_i to have the cavity circulating power E_{circ}. The number of light field E_i to be considered to have an accurate result depends on the finesse of the cavity.

6. The transmitted field E_{out} is simply the circulating field transmitted through the ETM.

In the above pseudo-code, we did not mention any resonance condition to maximize the circulating power in the cavity. Practically, we should always define the round-trip phase shift for the field in the cavity (or a microscopic position shift for the cavity length) before calculating the cavity circulating power. The round-trip phase shift allows us to set the cavity to be resonant for the fundamental mode or any other optical modes if necessary. The procedure to find the suitable resonance length is detailed in the next chapter in section 2.3.

1.5 Further reading

In this chapter, only a general and simplistic view of the FFT code has been presented. To have a better understanding of the power and limits of the code some further reading is highly recommended. Here some suggestions:

- The thesis from Brett Bochner[6], who programmed one of the first FFT code used by LIGO is a mine of treasures. The chapter two of this thesis about the technical realization of an optical FFT code contains some essential issues to grasp for a successful code, for example how the radius of curvature of the mirror can set an upper limit on the resolution of the grid, the use of aliasing filters or methods to calculate the steady state fields.

- Andrew Trigdell wrote also a FFT code in the ANU group. His master thesis may be hard to find but a following article[7] gives some good insights of the procedures involved in the code.

- A wonderful article to understand how to create a simple and robust FFT code to simulate high finesse cavity has been written by Partha Saha. The explanation on how to calculate the steady state cavity circulating field is crystal clear and very elegant (however the end of the article may appear a little bit more obscure at the first reading).

- Techniques to build a FFT code (and some other optical numerical codes) can be found in the recent thesis of Juri Agresti[8]. This thesis is also an excellent example of how to deal with non spherical mirrors.

- One of the first articles describing how to use a FFT to calculate the eigenmodes of a cavity by Gordon and Li[9]. Although the article is 40 years old, all the modern FFT programs are still based on the fundamental method explained in this paper.

Part I

OSCAR V1: the classic version

Chapter 2

The OSCAR code in details

This chapter is a step by step guide to run OSCAR **prior to version 3**. Using the simple example of a Fabry-Perot cavity, we will describe the essential procedures and give some hints in order to obtain valid results. The different scripts for this example can be found in the folder entitled Calculate_Pcirc. To run the full simulation, execute the Matlab script called Run_OSCAR.m.

To be able to run all the examples quoted in this chapter, an older version of OSCAR must be used: the version 1. However, this chapter is still there for legacy reason since all the principles described here are the basis for the current version 3. For completeness and to follow rigorously the examples, OSCAR V1.3 could be downloaded here [10].

2.1 The cavity to simulate

We are interested in simulating a Fabry-Perot cavity with a slightly mismatched input beam. The physical parameters of the optical system to simulate are summarized in table 2.1. After defining our cavity, we have to choose 2 essential parameters for our simulation: the physical size of the grid *Grid.Length* and the number of points of the grid *Grid.Num_point*. Our grid must include the mirror aperture, so the size of the grid must be at least equal to the mirror diameter. In our case the mirrors have a diameter of 25 cm, so the dimension of the grid could be 30 cm $\times$ 30 cm. Then, we can think about the number points required to sample the laser beam. Of course, a large number of points leads to accurate results but at the price of a lengthy computational time. From experience a grid with 128 $\times$ 128 is a safe choice to simulate a cavity with smooth mirrors.

Table 2.1: Parameters of the input beam and the Fabry-Perot cavity we wish to simulate. The variable name is the name of the variable in the OSCAR program and so also in the Matlab workspace.

Parameters		*Variable name*	Value
Cavity length	(m)	*Length_cav*	1000
Substrate refractive index		*Refrac_index*	1.5
Mirror diameter	(mm)	*Mirror.Diam*	250
Input laser			
Wavelength	(nm)	*Laser.lambda*	1064
Beam radius	(mm)	*Laser.size*	20
Wavefront curvature	(m)	*Laser.radius*	-2000
Optical power	(W)	*Laser.power*	1
Input mirror			
Radius of curvature	(m)	*ITM.RofC*	2500
Transmission		*ITM.T*	0.005
Loss	(ppm)	*ITM.L*	50
Reflectivity		*ITM.R*	1 - (Transmission + Loss)
End mirror			
Radius of curvature	(m)	*ETM.RofC*	2500
Transmission	(ppm)	*ETM.T*	50
Loss	(ppm)	*ETM.L*	50
Reflectivity		*ITM.R*	1 - (Transmission + Loss)

To speed up the calculations, we suppose the substrate of the input and end mirror to be thin, so the substrates are equivalent to thin lenses for a beam passing through. We also suppose the laser beam to be defined at the input mirror reflective coating but still outside the cavity, so we do not have to propagate the input laser beam in space or in the substrate before the transmission through the input mirror.

2.2 Declaration and initialization

The first Matlab script to run when starting a simulation is the script called CreateField.m. During this script all the variables required for the simulation will be defined. That includes as well the matrix representing all the mirror maps (in reflection and transmission), mirror aperture(s) and the matrix de-

scribing the input laser beam. For coherence, all the variables must be defined in the International System of Units (SI), which means that all the variables representing a length are in meters.

The script CreateField.m is most of the time self-explainatory and does not require extensive thinking. First the variables *Grid.Length* and *Grid.Num_point* are defined and then all the variables listed in the table 2.1 are given. From the given parameters, the mirror aperture matrix is created (according to the listing 1.4), following by the propagation matrix (listing 1.2), the mirror $\Delta OPL(x, y)$ maps (listing 1.3) and finally the matrix of the input beam.

For convenience, two matrices extensively used in intermediate calculations are also defined: *Grid.D2* is a matrix where the value of each pixel is the distance between the pixel and the origin, i.e $value = \sqrt{x^2 + y^2}$ and *Grid.D2_square* is the previous matrix but with every pixel squared, i.e $Grid.D2\_square = Grid.D2.^2$

2.3 Finding the resonance length

In this section the role of one of the most important script in OSCAR is explained. This script is called Find_resonance_length.m and is used to find the microscopic shift of the cavity length which is required to be on resonance. This script is essential because before calculating the circulating field in a Fabry-Perot cavity, the cavity has to be set on resonance where the circulating power of the fundamental mode TEM_{00} will be maximized. In the domain of gravitational wave detection, all the optical cavities of the detector are resonant for the fundamental mode or very near the resonance as in the case of DC-readout or detuned signal recycling.

2.3.1 Setting the resonance the length

The first thing to understand is how OSCAR implements a microscopic length shift of the cavity length. For example, for setting the cavity on resonance we have to shift the cavity length (called *Length_cav*) by a value δL with δL smaller than half a wavelength. The first (and the simplest) idea is to the set a new cavity length to *Length_cav*$+\delta L$. This solution is perfectly viable and gives correct result in Matlab for kilometer long cavities. However, from a numerical point of view it may not be the most robust solution since we have to add a length of the order of the kilometer with a length smaller than one

micrometer[1].

Another solution to set the cavity on resonance, is to add after each light round-trip in the cavity a constant phase shift. This is this solution that we use in OSCAR. So to simulate a cavity length shift of δL, a phase shift of $k2\delta l$ is added after each round-trip of the field E_i. Practically, when calculating the circulating field, the matrix of the field E_i is multiplied by a scalar factor $\exp(jk2\delta l)$ just before the field is reaching back the input mirror. To be consistent with the previous chapter it must have been $\exp(-jk2\delta l)$ however in the OSCAR code, it is implemented as $\exp(jk2\delta l)$, the sign convention can be arbitrary (as soon as it is kept constant for all the procedures).

In OSCAR, the variable which represents the shift in the cavity length necessary to be on the desired resonance is called *Length.reso_zoom*. To determine the right value for *Length.reso_zoom*, the script Find_resonance_length.m has to be run first. By convention, *Length.reso_zoom* is in fact $\delta l/2$ which means that the variable *Length.reso_zoom* represents the shift in the round-trip length to make the cavity resonant. With this convention if *Length.reso_zoom* is shifted by one wavelength, the resonance frequency is shifted by one free spectral range.

2.3.2 Finding the resonance length in details

The principle to find the resonance length of the cavity is quite simple: the cavity circulating power is monitored as the cavity round-trip length is scanned over one wavelength. The resonance length *Length.reso_zoom* is the length which maximizes the circulating power. A typical plot of the circulating power as a function of the microscopic cavity round-trip length is shown in figure 2.1.

The first idea to draw a plot of the cavity circulating power as a function of the cavity tuning is straightforward. We simply run the FFT code to calculate the circulating power for all the different de-tuning we would like to test. For example, in figure 2.1, the horizontal axis which spans over one wavelength is divided into 2000 points. So we can imagine to run the FFT code, 2000 times for each particular round-trip length. This procedure is absolutely correct, however extremely slow.

Practically, in OSCAR to draw the plot in figure 2.1, a technique first described by Gordon and Li[9] is employed. The principle is explained in the following steps :

[1]For reference, in Matlab the relative accuracy in the number representation can determined with the function eps('double'), on my computer it is of the order 10^{-16}.

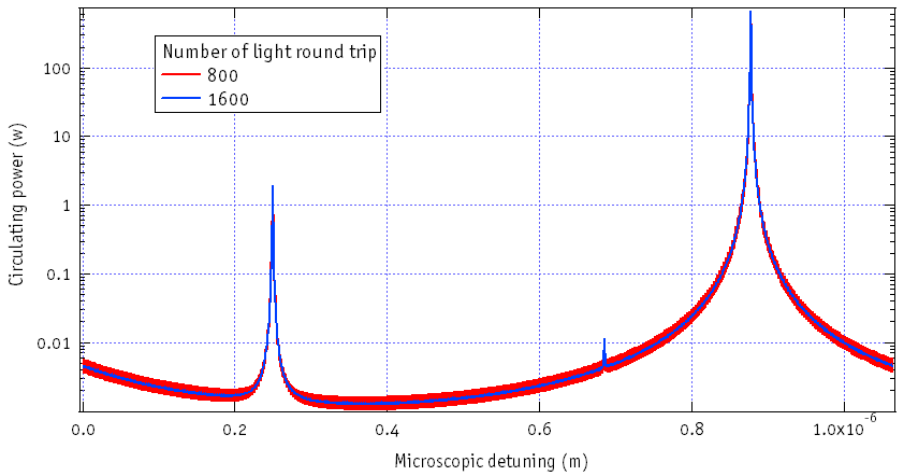


Figure 2.1: Circulating power in the cavity as the function of the macroscopic round-trip length de-tuning, scanned over one wavelength. The highest peak indicates the resonance position of the fundamental mode TEM_{00}. We can also notice two smaller peak revealing the presence of higher order optical modes ; this is a strong proof that the input beam is not matched with the cavity eigenmode. The figure has been plotted for two values of *Length.nb_iter*, this variable determines the number of fields E_i to include in the sum when calculating the circulating power. On resonance, the build up circulating power is of course higher when we use 1600 fields instead of 800 (even if it is not obvious in the figure due to the large vertical scale).

1. First all the fields E_i are calculated and stored for an arbitrary position of the cavity tuning (so for an arbitrary resonance length). This step is only done once.

2. The circulating field E_{circ} for a particular microscopic round-trip length δl is built up by summing all the field E_i with the proper phase shift:

$$E_{circ} = \sum_i E_i \exp\left(jk\delta l\right) \tag{2.1}$$

3. Repeat the previous step over all the de-tuning length δl we wish to try.

The advantage of this technique is that the FFT code is just called once to calculate the field E_i, then the reconstruction of the power buildup for the

various length de-tunings is just a matter of sums and multiplications. The only limitation of this technique is that a huge amount of memory is required since all the fields E_i are to be stored. For example, if we want to store 500 complex matrices of 256 $\times$ 256 points, 500 megabytes of free memory are required.

2.3.3 Code implementation

To calculate the resonance length in OSCAR, ie the value of *Length.reso_zoom*, several script are involved. Here the list:

- Find_resonance_length.m is the main procedure. At the end of the procedure, the variable *Length.reso_zoom* which maximizes the cavity circulating power, is returned. This script is divided into two similar parts: the cavity de-tuning is first scanned over one wavelength, then we scan around the maximum position (found after the first scan) with a much greater de-tuning resolution (we zoom around the maxima found in the first part). At the beginning of the script, two important variables are defined: *Length.nb_iter* determines the number of steps used to scan the cavity over one wavelength (usually 2000) and *Length.nb_propa_field* is the number of light round-trip taken into account when calculating the circulating power (i.e. number of fields E_i we used).

- Propagate_Field.m is a script called at the beginning of Find_resonance_length.m. This script calculates all the intermediate field E_i using the FFT code and stores the results in one variable called *Field.propa*. *Field.propa* is a 3D matrix having with a size of *Grid.Num_point* $\times$ *Grid.Num_point* $\times$ *Length.nb_propa_field*.

- Build_Field_Cavity.m is a function called intensively by Find_resonance_length.m. This function takes for argument a length de-tuning and returns the build-up circulating field following the equation 2.1.

- Calculate_power.m is a simple function which takes for input a 2D electric field and returns the optical power in Watt of the input field.

2.3.4 Some comments

The method used by OSCAR to find the resonance length is slow. In terms of calculation time, it is the bottleneck of this FFT code. It is possible to find different approaches to calculate the resonance length and some are much

faster, however I prefer to keep the method described above. Why ? Because the plot of the circulating power as a function of the wavelength (figure 2.1) contains much essential information which help debugging the simulation or understanding the optical system. Here are some examples :

- If during a simulation, the plot of the circulating power as a function of the de-tuning does not look like the one in figure 2.1, but instead looks flat or with very small bumps it means the cavity is unstable. In the same idea, the line-width of the peak in the plot is inversely proportional to the finesse of the cavity, so gives a good indication of the round-trip loss in the cavity.
- The number of peaks in the plot is directly proportional to the mode-mismatching or misalignment between the input beam and the cavity eigenmodes. In the case of perfect mode-matching only one peak is present, which means that all the input light is coupled to only one optical mode (preferably the fundamental Gaussian beam). By looking at the shape of the higher order modes that are excited (see figure 2.1), we could have an idea of the type of mode-mismatching and/or misalignment. For example, if all the higher order modes that are excited look like TEM_{m0}, it means the input beam is misaligned with the cavity axis in the horizontal direction.
- Finally, since we also know the position de-tuning for the higher order modes, we can also set the cavity on the resonance of the higher order modes if necessary. The knowledge of the relative de-tuning position of the resonance for the higher order modes allows also the calculation of the Gouy phase shift between higher order modes (which maybe unknown if the beam is not Gaussian).

We do not need many of light round-trip to have an accurate result for the resonance length position. In the previous example, we use 800 or 1600 round-trips for the calculation but only 50 round-trips can already give a correct answer. The only difference is that the plot of the circulating power as a function of the de-tuning may not look so sharp (so we can miss the presence of smaller resonance peaks).

For verification purpose, it may be important to check the shape of the circulating field for different de-tuning lengths. For example to display the circulating field for a de-tuning position of 2.5×10^{-7}, which corresponds to

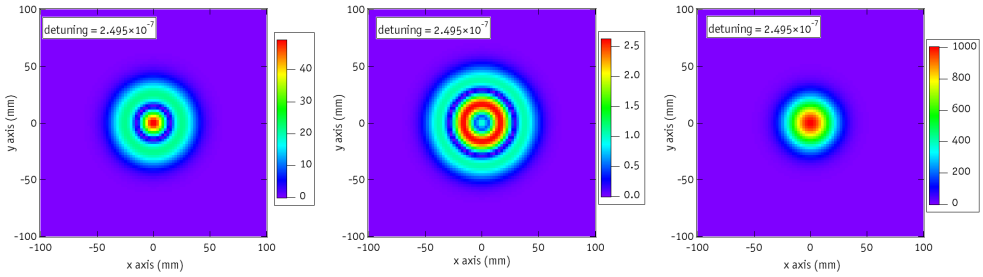


Figure 2.2: 2D amplitude of the circulating field responsible for the 3 peaks in the plot of the cavity circulating as a function of the de-tuning (figure 2.1). We can recognize from left to right the mode LG_{10}, LG_{20} and finally the fundamental mode LG_{00}. The presence of the 3 first LG_{m0} optical modes indicates that the input beam is properly aligned with the cavity axis but that we have a slight mode-mismatching.

the first left peak in the figure 2.1, we just have to write the following command in Matlab:

>> Plot_Field(Build_Field_Cavity(2.5E-7))

With such a command, we can check which optical modes can build up in the cavity. The 2D amplitude of the three optical modes corresponding to the three peaks in figure 2.1 are presented in figure 2.2 in the order of increasing de-tuning.

2.4 Calculating the circulating field

The previous procedure Find_resonance_length.m is essential to determine the operating point for the cavity. In most cases the cavity is locked on the fundamental mode TEM_{00}, so the procedure Find_resonance_length.m returns by default the value of the de-tuning required to make the TEM_{00} resonant inside the cavity. It is always implicitly assumed that the maximum circulating power is obtained for a resonant TEM_{00} and not for a higher order optical mode.

After we have calculated the resonance length, the script Get_results.m can be called. This script calculates the static fields in the Fabry-Perot cavity

and displayed the total circulating power as well as other results.

The procedure to calculate the circulating field was explained in the section 1.4 and is briefly reminded here. First the input laser field (variable *Field.Start* in OSCAR) crosses the input test mass substrate, creating the intermediate circulating field (variable *Field.Circ* and also called E_1 in figure 1.6). Then the intermediate circulating field is propagated back and forth between the cavity mirrors with the number of round-trips determined by the variable *Iter.final*. The total circulating field (variable *Field.Total*) is derived by summing all the intermediate fields at a defined position in the cavity, in OSCAR it is done after the reflection from the input mirror. Everything described above can be simply achieved in Matlab as shown in the listing 2.1.

Listing 2.1: The core of the OSCAR program to calculate the circulating field in a cavity

```
Length.reso_zoom = 8.7794200e-007;
Iter.final = 3000;

Phase_shift = exp(i*Laser.k_prop* Length.reso_zoom);

Field.Circ = Propa_mirror(Field.Start, Mirror.ITM_trans,i*ITM.t);

for q = 1:Iter.final

    Field.Total = Field.Total + Field.Circ;

    Field.Circ = Make_propagation(Field.Circ,Mat_propagation);
    Field.Circ = Propa_mirror(Field.Circ,Mirror.ITM_cav,ETM.r);
    Field.Circ = Make_propagation(Field.Circ,Mat_propagation);
    Field.Circ = Field.Circ * Phase_shift;
    Field.Circ = Propa_mirror(Field.Circ, Mirror.ITM_cav,ITM.r);

end
```

After the calculation of the total circulating field in the cavity *Field.Total*, the transmitted beam *Field.Transmit* and the reflected beam *Field.Reflect* can be easily derived. The transmitted beam is simply the total circulating field after a transmission through the end mirror. The reflected beam is the sum of the input field directly reflected by the input mirror and the field leaking from the cavity (which is the total circulating field transmitted by the input mirror).

2.5 Displaying the results

After the total circulating, transmitted and reflected fields have been calculated, the results can be displayed. First some parameters (with obvious names) are written in the Matlab command window, then a 2D plot of the different optical fields present is displayed as shown in figure 2.3.

```
------ Display the results ------
Circulating power (W): 749.256602
Reflected power (W): 0.885891
Transmitted power (W): 0.037463
Beam radius on the ITM (m): 0.020574
Beam radius on the ETM (m): 0.020577
Cavity waist size (m): 0.016462
Location of the waist from ITM (m): -499.853070
```

To feel confident in the results from OSCAR, it is always good to test the results with different choices of number of light round-trip or size of the grid. If the results are very sensitive to one parameters, it usually means that there is a problem somewhere, and further checks are required.

For example, the results as a function of number of light round-trip are presented in the table 2.2. As expected, on resonance, the circulating power (defined by the power of the sum of all the fields E_i) increases when the number of field E_i computed increases. The right number of iterations to consider for a simulation depends on the finesse of the cavity. For a low finesse cavity, with a high round-trip loss, the power contained in the field E_i will quickly decrease and rapidly becomes negligible after few round-trips. In table 2.2, the FFT results are compared with the results from the software Finesse [11], which is based on mode expansion. The Finesse script used to simulate the Fabry-Perot cavity is presented in appendix B.

Another important parameter to test is the resolution of the grid. The results for different sizes of the grid are presented in the table 2.3. As we can see, the results are pretty robust even with a coarse grid. For a grid size of 32×32, the grid resolution is 1 cm, which means the beam radius is just represented by 2 pixels in the cavity. Even with such a low resolution, the results are still accurate.

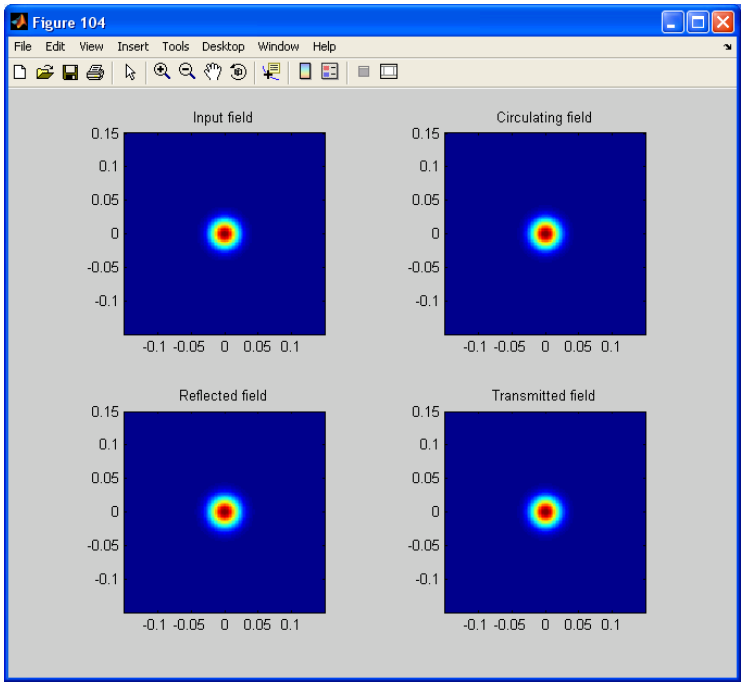


Figure 2.3: Amplitude profile of the different steady state optical fields in the Fabry-Perot cavity.

Table 2.2: Influence of the total number of light round-trip considered (variable *Iter.final*) on the results. The size of the grid is set to 128 × 128.

Number of iteration	Circulating power (W)	Cavity waist size (mm)	Cavity waist position from IM (m)
1000	640.712	18.403	-499.835
2500	747.546	18.403	-499.852
5000	749.903	18.403	-499.853
10000	749.906	18.403	-499.835
Finesse	749.906	18.403	-500

Table 2.3: Influence of the size of the grid (variable *Grid.Num_point*) on the results. The number of iteration is 5000. The computation time is normalized by the computation time required for a grid of size 128 × 128 (which is less than 2 minutes on a modern computer).

Size of the grid	Circulating power (W)	Cavity waist size (mm)	Cavity waist position from IM (m)	Normalized computation time
32 × 32	749.902	18.403	-499.854	0.09
64 × 64	749.903	18.403	-499.853	0.31
128 × 128	749.903	18.403	-499.853	1.00
256 × 256	749.902	18.403	-499.844	7.32
Finesse	749.906	18.403	-500	-

2.6 Calculating the cavity eigenmodes and diffraction loss

We can first wonder what is called a cavity eigenmode ? A cavity eigenmode is an electric field which comes back exactly with the same spatial profile after one cavity round-trip. The spatial profile must be identical but not necessary the amplitude since the cavity may be lossy. The usual cavity eigenmodes are the set of Hermite-Gauss and Laguerre-Gauss for cavity with spherical mirrors, as most of the readers must already know.

For high finesse cavities[2], the circulating field is in fact a cavity eigenmode. This can be easily understood if the input cavity field is negligible compared to the cavity circulating field, a more rigorous demonstration can be found in [12] for example. Mode cleaner cavities are based on this principle, the transmitted beam is the TEM_{00} cavity eigenmode, whereas the input beam may be composed of several optical modes.

Usually, we seldom need a FFT code to calculate the eigenmodes of a Fabry-Perot cavity with spherical mirrors since the exact analytical solutions are known [13]. However, we may want to know the cavity eigenmodes if the mirrors are not perfectly spherical (because of thermal lensing) or if beam clipping due to finite-size mirrors is important. To know the cavity eigenmodes in this case, we defined an input beam close in shape to the supposed cavity eigenmodes and then calculate the cavity circulating field. The cavity circulating field will be the cavity eigenmode. Some problems may arise if

[2]Quantitatively (and approximately), we could say that a cavity has a high finesse when the the circulating power is much higher than the input power

the cavity is nearly degenerate and the input beam is composed of several resonant or near resonant modes. In this case we can think of two solutions :

1. Increase the cavity finesse to have a better separation between optical modes and if necessary generate some specific losses to attenuate the undesirable optical modes.

2. Take the cavity circulating field as a new input field and start a new calculation. This step can be done several times and could be understood as cascading optical cavities to increase the mode-cleaning effect.

Since we have now an idea of how to calculate a cavity eigenmode, we can also try to calculate the diffraction loss for this mode. We call here "diffraction loss" the loss due to the finite size of the mirrors. This loss is also sometime referred as "clipping loss". Since in theory, Gaussian beams have an infinite spatial extent, diffraction losses are always present. However the loss can become negligible for large diameter mirrors and small laser beam radius. For example, the diffraction loss of a laser beam of beam radius 6 cm after reflection on a 15 cm radius mirror is only 4 ppm [14]. The diffraction losses are more important for higher order optical modes and ultimately can even affect the profile of the mode [15].

In OSCAR the diffraction loss of an eigenmode is computed by calculating the round-trip loss of the mode when the reflectivities of the cavity mirrors are set to 1. So the first step is to calculate the cavity eigenmode, this is usually done by calculating the circulating field in a high finesse cavity. The second step is to normalize the power of the eigenmode to 1 and then propagate the mode one round-trip in the cavity whose mirrors reflectivities are set to 1. Finally, the diffraction loss is simply the power lost by the mode during one round-trip. The above steps are shown in the listing 2.2.

An example of how to calculate diffraction loss with OSCAR can be found in the folder called Calculate_diffraction_loss. This example is detailed in the chapter 3.2 of this manual.

2.7 A typical OSCAR run

A typical run of OSCAR consists of running consecutively three different Matlab scripts:

1. CreateField.m is the script used to initialize the variables and define the optical cavity parameters, the mirror maps as well as the input beam.

Listing 2.2: Piece of code used to calculate the diffraction loss. We suppose that we have already calculated the circulating field *Field.Total*.

```
% Normalise the circulating field (=the eigenmode)
Field.loss = Field.Total;
Field.loss = Field.loss/sqrt(Calculate_power(Field.loss));

% Make a round-trip with a reflectivity of 1 for the mirrors
Field.loss = Make_propagation(Field.loss,Mat_propagation);
Field.loss = Propa_mirror(Field.loss,Mirror.ETM_cav,1);
Field.loss = Make_propagation(Field.loss,Mat_propagation);
Field.loss = Propa_mirror(Field.loss, Mirror.ITM_cav,1);

% Calculate the diffraction loss
Dif_loss = (1 - Calculate_power(Field.loss));
fprintf('Diffraction loss per round-trip: %d \n',Dif_loss);
```

2. Find_resonance_length.m is used to find automatically the resonance length of the cavity which maximizes the circulating power. In most cases, the resonance length locks the cavity on the TEM_{00}. Manually it is possible to lock the cavity on any arbitrary position, for example to see the resonance of one higher order mode.

3. Get_results.m is the main procedure which gets the circulating field in the cavity and, if required, also the reflected and transmitted fields as well as the diffraction loss of the circulating field.

The calls for the three previous script are usually reunited into one single scrip called Run_OSCAR.m.

2.8 Script and function list

Here is the list of Matlab scripts that you may find in every OSCAR folder. Depending on the goal of the simulation, the scripts may not be exactly the same and some variations exist.

- Beam_parameter.m is a function which takes for parameter a complex 2D Gaussian field and returns the beam radius and the wavefront radius of curvature. The fit only works for fundamental Gaussian beam but it can easily be adapted to also fit higher order modes.

- Build_Field_Cavity.m is a function used to find the resonance length, see section 2.3.3.

- Calculate_power.m is a simple function which takes a 2D field and returns the optical power of the field.
- CreateField.m is the first script called in OSCAR to initialize all the variables, see section 2.2.
- CreateMirror.m is the script to create the matrix representing the wavefront distortion induced by the mirrors, see section 1.3.2.
- Find_resonance_length.m is used to find the microscopic de-tuning required to set the cavity on resonance, see section 2.3.2.
- Get_results.m is the main OSCAR procedure to calculate the circulating power in the cavity. The script is also used to do some post processing of the results, see section 2.4 and 2.5.
- Make_propagation.m propagates a 2D field over a certain distance. The function takes two arguments (a field and a propagation matrix) and returns the field after propagation. The function is the implementation of the last three lines in listing 1.2.
- Plot_Field.m is a function which takes a 2D field and plots the amplitude of the field in 2D (or 3D if desired).
- Propa_mirror.m is a function to simulate the effect of a wavefront distortion. The function takes a 2D input field, a mirror map or any distortion and a reflectivity (or transmittivity) and returns the field after reflection (or transmission). The function is described in section 1.3.5.
- Propagate_Field.m creates the initial 3D matrix used to calculate the circulating field for the different de-tuning, see section 2.3.3 for further explanations.

Chapter 3

Applications

In this chapter, we provide examples of some typical results that can be obtained with OSCAR. These examples could be taken as a starting point to build more complex simulations. The OSCAR scripts associated with each example are provided in the OSCAR package. The example described here are using the classic version of OSCAR (version 1.X) which can be downloaded here [10]. **For new users, it is recommended to use OSCAR V3.x and so jump directly to the section 5**

3.1 Distortion of the optical field due to thermal lensing

Let us consider the same Fabry-Perot cavity as the one described in the previous chapter (section 2.1). Instead of using an input laser beam of 1 W, we upgrade the input power to 500 W (so the circulating power is now 375 kW) and we are interested in simulating some thermal lensing. Both input and end mirrors are supposed to be made of fused silica with a substrate absorption of 2 ppm/cm and a coating absorption of 0.5 ppm.

At least two distinctive effects can be induced due to the optical power absorbed in the mirrors:

- A temperature gradient inside the substrates of the mirrors appears. The temperature gradient generates a refractive index gradient (thermo-optic effect) which induces a wavefront distortion for the beam crossing the optics.
- Since the temperature is no longer uniform in the test mass, the curvature of the optic surface will change as a result of thermal expansion. In

this case we can expect the eigenmode of the cavity to change as well.

Before we continue, it is better to make some assumptions to lighten the calculations. If the reader understands the method described here it is straightforward to implement a full model. First we will suppose that we have uniform absorption in the test mass, so we can take advantage of the cylindrical symmetry. The power absorbed is dominated by the coating absorption, so we will only take into account the temperature gradient $T(r,z)$ in the optics and the thermal expansion of the mirror high reflective (HR) coating surface $\delta s(r)$. The deformations of the anti reflective coating are assumed to be negligible.

OSCAR cannot simulate directly the effect of the optical absorption. If required an analytical formula can be implemented [16] however it is not as flexible as finite-element simulations in particular if thermal lensing compensation schemes have to be investigated. I used the software ANSYS to simulate the temperature gradient $T(r,z)$ in the optics and the thermal expansion of the mirror surface $\delta s(r)$[1]. A schematic of the hot cavity is shown in figure 3.1.

From the temperature distribution inside the substrate and the sagitta change, we can derive the optical path length difference $\Delta OPN_{sub}^{trans}(r)$ induced by the substrate in transmission according to:

$$\begin{aligned}\Delta OPN_{sub}^{trans}(r) &= \int_0^L n(r,z)dz - \int_0^L n(0,z)dz + (n-1)\delta s(r) \\ &= \int_0^L \beta T(r,z)dz - \int_0^L \beta T(0,z) + (n-1)\delta s(r)dz \end{aligned} \quad (3.1)$$

With β the thermo-optic coefficient and n the refractive index of the substrate. Similarly, we can calculate the wavefront distortion $\Delta OPN_{sub}^{ref}(r)$ for the input beam reflected directly reflected on the input mirror:

$$\Delta OPN_{sub}^{ref}(r) = 2\left(\int_0^L \beta T(r,z)dz - \int_0^L \beta T(0,z) + n\delta s(r)\right) \quad (3.2)$$

And finally, the wavefront distortion of the beam reflected on the mirrors inside the cavity is:

$$\Delta OPN_{cav}^{ref}(r) = 2\delta s(r) \quad (3.3)$$

[1]We assume uniform absorption in the test mass, so we can take advantage of the cylindrical symmetry.

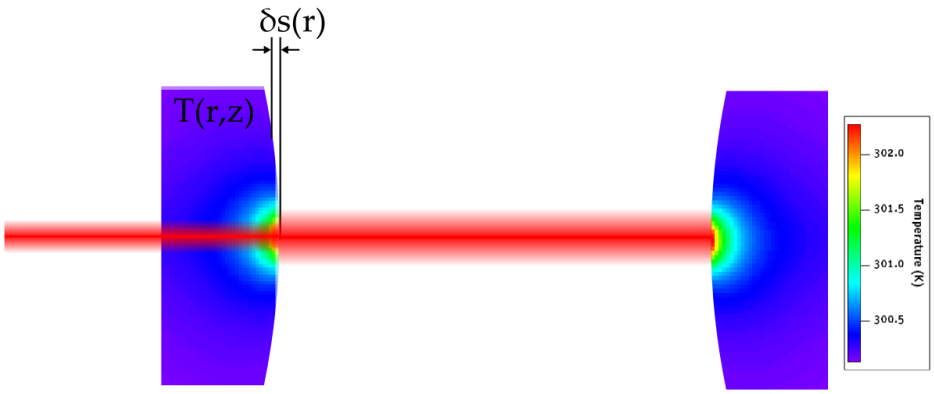


Figure 3.1: Schematics of the hot cavity. The mirror diameter is 250 mm and the thickness is 100 mm. We can notice that the temperature profile is dominated by the coating absorption. The optical parameters are detailed in the text. For clarity, we did not represent the curvature of the cold optics and we suppose the same distortion in the input and end mirrors. $T(r, z)$ represents the temperature distribution inside the test mass and $\delta s(r)$ the change in sagitta due to the optical absorption.

Of course, the three wavefront distortions just defined have to be added to any wavefront distortion already present when the cavity is cold, especially the ones induced by the curvature of the mirrors. For references, the main distortions due to thermal lensing which have to be included in OSCAR are plotted in figure 3.2.

Here an example how to proceed in reality. First run ANSYS to simulate the temperature distribution in the test mass as well as the thermal expansion of the optic. Then from these results, we can save a text file with two results, first the wavefront distortion induced by the thermo-optic effect for the optic in transmission and the change in sagitta of the optics. The text file is then loaded in OSCAR and the new wavefront distortions for the all optics of the cavity are calculated accordingly. An example of such a text file is presented below, the first column is the radius, the second the optical path length difference in transmission and the third the change in sagitta, all the columns are in meters.

```
-1.2500000e-001 -1.0824598e-006 -2.6695000e-008
```

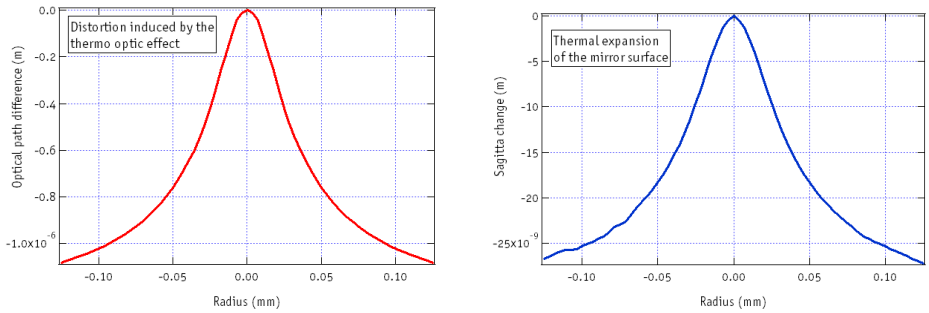


Figure 3.2: Optical path length difference induced by the mirror substrates in transmission when only the thermo optics effect is considered (left) and change in the sagitta of the reflective side of the mirrors (right). It interesting to note the difference between the two vertical scales, the left plot represents an optical path in micrometers, whereas the right plot is in tens of nanometers. These plots are derived from the results of ANSYS simulations and will then be integrated into OSCAR.

```
-1.2142857e-001 -1.0731224e-006 -2.6433549e-008
-1.1785714e-001 -1.0655394e-006 -2.6172131e-008
-1.1428571e-001 -1.0577238e-006 -2.5911061e-008
-1.1071429e-001 -1.0496365e-006 -2.5695000e-008
-1.0714286e-001 -1.0398661e-006 -2.5695000e-008
-1.0357143e-001 -1.0304229e-006 -2.5615955e-008
-1.0000000e-001 -1.0216618e-006 -2.5349271e-008
```

The file can found in the OSCAR distribution under the name From_ANSYS.txt in the folder Calculate_TL_effect. A graphical representation of the data file from ANSYS is shown in figure 3.2.

Most of the time, the resolution used in the results file from ANSYS is different from the grid resolution used in OSCAR, so we have to resample the results using an interpolation method. How the results file is integrated into OSCAR is presented into the listing 3.1

After implementing the distorted mirrors, we can calculate the resonance length for the fundamental mode and then calculate the total circulating field. This is done by running successively the scripts Find_resonance_length.m and Get_results.m. The comparison between the cold cavity is presented in table 3.1.

Listing 3.1: Commands used to read thermal lensing results from ANSYS.

```
% Load the results file
load('From_ANSYS.txt')
loaded.radius = From_ANSYS(:,1);
loaded.TL = interp1(From_ANSYS(:,1),From_ANSYS(:,2),Grid.D2,'spline')*0.2;
loaded.sag = interp1(From_ANSYS(:,1),From_ANSYS(:,3),Grid.D2,'spline')*0;

% Add the thermal lensing distortion to the previous wavefront
% Special attention to the sign!

Mirror.ITM_cav = Mirror.ITM_cav - 2*loaded.sag;
Mirror.ETM_cav = Mirror.ETM_cav - 2*loaded.sag;

Mirror.ETM_trans = Mirror.ETM_trans + loaded.TL - (Refrac_index-1)*loaded.sag;
Mirror.ITM_trans = Mirror.ITM_trans + loaded.TL - (Refrac_index-1)*loaded.sag;

Mirror.ITM_ref = Mirror.ITM_ref + 2*loaded.TL - 2*Refrac_index*loaded.sag;
```

Table 3.1: Comparison of the cavity gain and size of the beam on the input mirror for a cavity with and without thermal lensing.

	Cold cavity	Hot cavity
Cavity gain	749.3	444.8
Beam radius on IM (mm)	20.6	20.8

Some very interesting points can be deduced by understanding the two lines of the table 3.1:

- Due to thermal lensing, the beam radius only increases by 1 %. That indicates that the mirror profiles are only slightly affected by thermal lensing since the cavity eigenmodes are very similar for both cases: cold and hot cavities. This is not a surprise since the change in sagitta of the reflective sides of the mirrors is relatively small as we have previously seen.

- The decrease in the optical gain is quite important, since the optical gain is almost divided by a factor 2 between the cold and hot cases. It means in the hot cavity case, we have a strong mode mismatching between the input beam and the cavity fundamental mode. Since the latter has almost not changed, we can deduce that the mode mismatching is induced by the thermal lens in the substrate of the input mirror.

- We have assumed a certain amount of optical power absorbed in the mirror, it was based on the optical power circulating in the cold cavity. But since the circulating has decreased due to the mode mismatching, our thermal lens calculation is wrong. A simple iterative process can be written to solve this problem and find the steady state parameters.

- This simple example shows that in our fused silica mirrors, the main thermal lensing effect is due to the thermal lens in the substrate of the input mirror generated by the optical absorption in the high reflective coating. The conclusion may be different for different substrates such as sapphire or calcium fluoride.

3.2 Calculating diffraction losses

One of the most promising application of FFT optical codes is to calculate the diffraction loss of the circulating cavity field. The method used to calculate the diffraction loss has been explained in section 2.6. An example how to calculate the diffraction loss of the mode HG_{10} is presented in the folder Calculate_diffraction_loss.

The diffraction loss calculation can be decomposed into three steps:

1. Find the resonance length of the cavity for the mode HG_{10}. To do this, we could inject a mode HG_{10} and maximize the circulating power. This approach is however not so easy : instead we will inject a uniform pattern of light and excite all the optical modes HG_{m0}. Then we will select manually the resonance length of the mode HG_{10} in the spectrum of the cavity.

2. Find the HG_{10} eigenmode of the cavity. The shape of this mode can be different from the theory if the mode undergoes some serious clipping.

3. Calculate the diffraction loss as the power lost during one light round-trip with perfectly reflective mirrors.

We decide to not inject directly a mode HG_{10} (right plot on figure 3.3) but instead inject a simpler light pattern which will excite all the optical modes along the horizontal axis. This pattern is one vertical strip with positive amplitude and one vertical strip with negative amplitude as shown in the left plot figure 3.3.

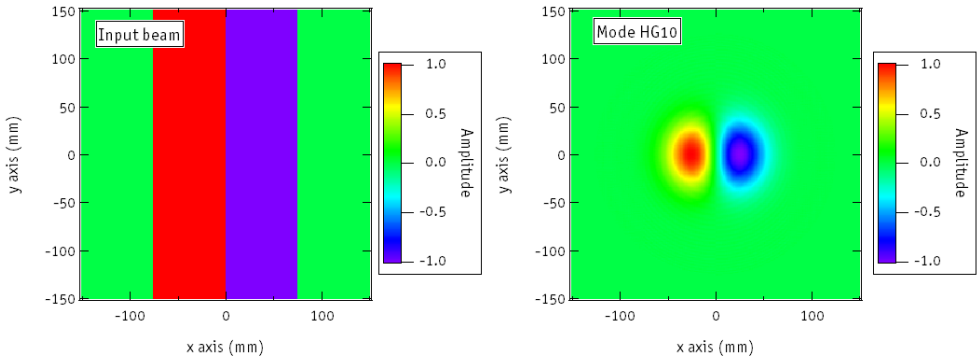


Figure 3.3: In the left plot, is the profile of the input electric field that we inject in the cavity to excite the HG_{10} shown on the right. Both optical pattern are normalized in amplitude.

Table 3.2: Influence of the size of the grid on the diffraction loss).

Size of the grid	Diffraction loss (ppm)
128 × 128	618
256 × 256	620
512 × 512	621

By using the procedure Find_resonance_length.m, we can calculate all the cavity modes excited due to our particular input beam. The circulating power as the cavity is scanned over one FSR is shown in figure 3.4. Each peak in this plot represents the resonance of one of the cavity mode. We can display different cavity eigenmodes for the first four highest peaks and we found that the mode HG_{10} resonates for a de-tuning of 3.181×10^{-7} m.

Since we have found the resonance length for the mode HG_{10}, we can now use the procedure Get_results.m to calculate the diffraction loss using the method described in section 2.6. As a result, we found the diffraction loss equal to 618 ppm per round-trip. Important beam clipping can also change the shape of the eigenmode of the cavity, as can be easily demonstrated using a FFT code[15].

We can now check how the diffraction loss depends of the size of the grid. The results are shown in 3.2. As we can see the results do not change as we increase the size of the grid (which is a good sign). Below a grid size of 128 × 128, no realistic eigenmodes can be found in the cavity.

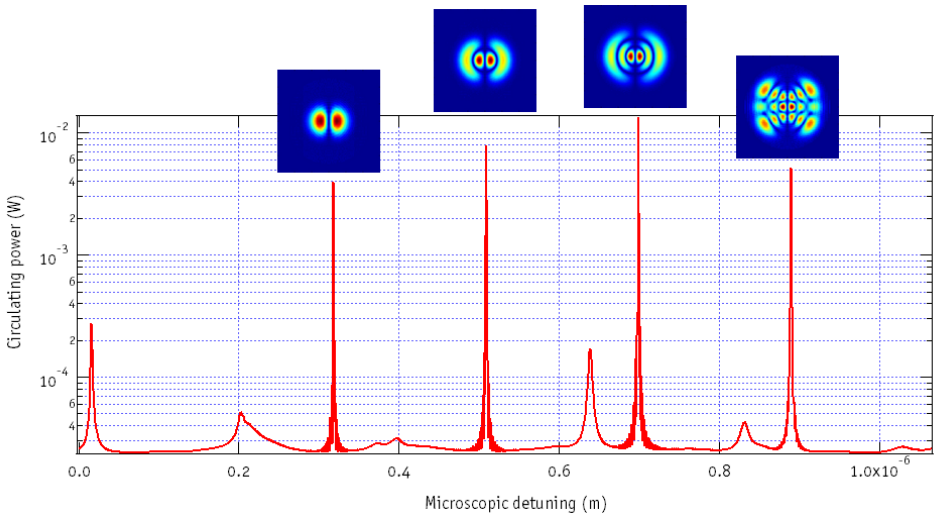


Figure 3.4: Cavity spectrum showing the different resonance lengths of the optical modes. The power profile of the main optical modes is presented above their respective resonance peaks.

3.3 Using flat beams

In this section, we will show how OSCAR can be used to calculate the frequency separation between higher order modes. This is in fact an indirect calculation of the Gouy phase shift between optical modes. To make the example more interesting, we will not use the usual Gaussian beams but flat beams. As a consequence of using flat beams, the mirrors of our cavity will no longer have spherical profiles [17].

To create the mirror profiles, we will not use directly the analytical formula which is relatively complicated, but instead we will use the wavefront curvature of the flat beam at the mirror position. Since the flat beam is an eigenmode of the cavity, the curvature of the mirrors must match the wavefront of the incoming beam. So the three steps to calculate will be:

1. Use the simple analytical formula to define the flat beam at the waist of the cavity. Since both mirror will have identical profile, we know that the waist position is in the middle of the cavity.

Listing 3.2: Script to create the mirror profiles used to support a given light field

```
%---------------------- Create nearly concentric flat beam --------------------
% Create the profile at the cavity waist

waist_0 = sqrt((Laser.lambda * Length_cav) / (2*pi));
p = 3*waist_0; % Defined the width of the Mesa beam

x_temp = Grid.D2/waist_0;
Field.mesa_tmp = (1./x_temp).*exp(-x_temp.^2).*besselj(1,2*x_temp*p/waist_0);

clear('x_temp')

% Propagate the beam from the waist to the mirror
Field.On_mirror = Make_propagation(Field.mesa_tmp,Mat_propagation_h);

% Take the phase of the beam at the mirror and calculate the equivalent
% change in sagitta
Mirror.ETM_cav = (2/Laser.k_prop)*angle(Field.On_mirror);
Mirror.ITM_cav = Mirror.ETM_cav;
```

2. Propagate the beam along half the cavity length, so that the flat beam is now at the mirror position.

3. Calculate the wavefront curvature of the beam at the mirror and then set the mirror profile to be identical to the wavefront.

The method described in the three points above can be directly implemented in OSCAR as shown in the listing 3.2.

Since we are just interested in the frequency difference between 2 eigenmodes of the cavity (and not in the cavity circulating power), the input laser beam only needs to be slightly matched to the cavity eigenmodes (and we must have a high finesse to achieve a good mode selection). The important point is that the input beam couples to (or excites) at least the two cavity eigenmodes of interest.

As for the input light field, we decided to use a classic fundamental Gaussian beam. By this way we will couple to most the cavity eigenmodes which are circularly symmetric, e.g. the equivalent of the LG_{m0} modes. For example the spectrum of the cavity as we scan over one wavelength is shown in figure 3.5. As expected we managed to excite the fundamental mode and also some other higher order modes.

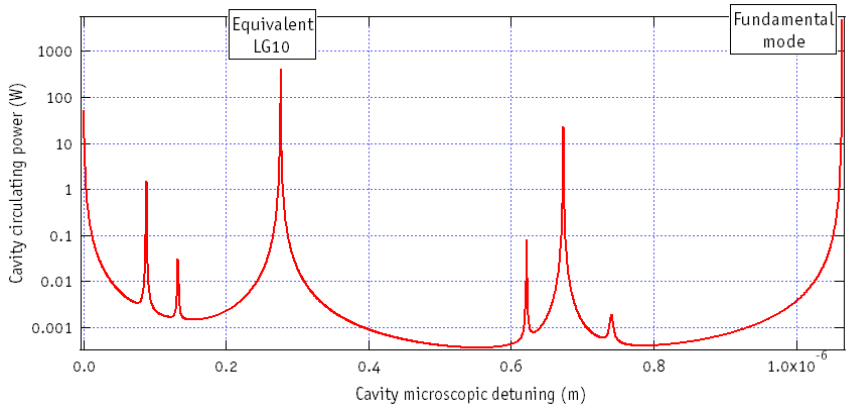


Figure 3.5: Cavity spectrum of the cavity supporting flat beams. It is important to note the fundamental for a de-tuning of 0. This is not a coincidence, but a direct consequence of the way we defined the mirror profiles (by propagating a beam from the cavity waist).

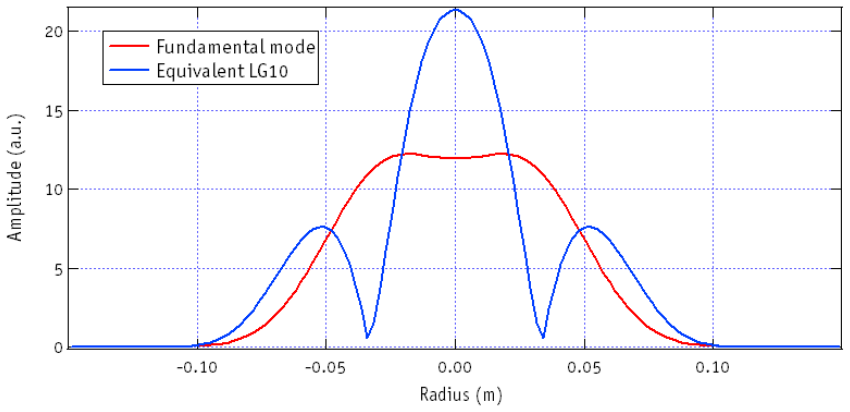


Figure 3.6: Comparison of the profile of the fundamental flat beam with the equivalent LG_{10}. Both beam contains the same optical power.

Just to have a look, we can display a cross section of the fundamental flat beams and the equivalent LG_{10} to compare the energy distribution. Such a comparison is shown in figure 3.6.

We can thus go back to our initial question : what is the frequency separation between the fundamental mesa beam and the mode LG_{10} ? From the spectrum plot, we know the difference of de-tuning in length between the fundamental mode and the equivalent LG_{10} is $\Delta l = 2.7664 \times 10^{-7}$ m. Since the displacement of one wavelength λ is equivalent to one free spectral range $(c/2L)$, the frequency difference Δf between the 2 modes is simply:

$$\begin{aligned} \Delta f &= (c/2L) * (\Delta l/\lambda) \\ &= (3 \times 10^8/4000) * (2.7664 \times 10^{-7}/1.064 \times 10^{-6}) \\ &= 19.5\text{kHz} \end{aligned} \tag{3.4}$$

Using a similar calculation, we could also deduce the Gouy phase shift between the two cavity eigenmodes.

3.4 Deriving a Pound-Drever-Hall locking signal

In this example, we will show how to implement sidebands in OSCAR. For testing purpose, we can try to derive the Pound-Drever-Hall (PDH) error signal which is used to lock a Fabry-Perot cavity on resonance [18]. The error signal is derived in reflection by combining a resonant (or near resonant) carrier field with a non resonant pair of sidebands. In this case, it can be shown that the error signal is in fact proportional to the imaginary part of the reflected field [19]. The script for this example can be found in the folder Calculate_PDH_signals

So the first thing to do, is to create sidebands using a phase modulator. Consider an electric field of amplitude E_0 and angular frequency ω_0 incoming on a phase modulator. E_m the electric field after the modulator can be written as:

$$\begin{aligned} E_m &= E_0 \exp(i\omega_0 t + m \sin(\omega_m t)) \\ &= E_0 \exp(i\omega_0 t) \sum_{k=-\infty}^{+\infty} (-1)^k J_k(m) \exp(ik\omega_m t) \end{aligned} \tag{3.5}$$

With m the modulating index and ω_m the modulation frequency. $J_k(m)$ is the Bessel function of the first kind of order k. For example, the values for the first Bessel functions are shown in table 3.3. As a good approximation we can just consider the first order sidebands $k = \pm 1$. So equation 3.5 can be simplified to:

Table 3.3: Values of the first Bessel functions for different modulating indexes m. Conveniently, the Bessel functions follow the property $J_{-k}(m) = (-1)^k J_k(m)$

m	0.1	0.2	0.4	Taylor series
J_0(m)	0.9975	0.9900	0.9604	$1 - \frac{m^2}{4} + \mathcal{O}(x^4)$
J_1(m)	0.0499	0.0995	0.1960	$\frac{m}{2} + \mathcal{O}(x^3)$
J_2(m)	0.0012	0.0050	0.0197	$\frac{m^2}{8} + \mathcal{O}(x^4)$
J_3(m)	0.0000	0.0002	0.0013	$\frac{m^3}{48} + \mathcal{O}(x^5)$

$$E_m = E_0 \exp(i\omega_0 t)(1 + \frac{m}{2}\exp(i\omega_m t) - \frac{m}{2}\exp(-i\omega_m t)) \tag{3.6}$$

So the electric field after the phase modulator can be expanded as the superposition of three electric fields of angular frequency ω_0 (the carrier) and $\omega_0 + \omega_m$ and $\omega_0 - \omega_m$ (respectively the upper and lower sidebands). Since each field has different frequencies, they will also have different resonant conditions in the cavity.

Consider the sideband with the frequency $\omega_0 + \omega_m$. Propagating over a distance L the sideband undergoes the phase shift $\exp(i\frac{\omega_0+\omega_m}{c}L)$. That means, the sideband phase shift is the phase shift of the carrier plus an additional phase shift. Since before entering the cavity, the carrier and the sidebands have the same spatial profile, we can simulate the sidebands in the cavity by taking the simulated carrier and by adding an extra phase shift.

In practice, to simulate the carrier and the sidebands, we can simulate independently three optical fields, the carrier and the lower and upper sidebands. The three fields can be treated the same way in the simulation, except that they have different round-trip phase shifts. Below is the three phase shifts for one round-trip in the cavity:

$$\begin{aligned}
&\exp(ikLength.reso_zoom) && \text{for the carrier} \\
&\exp(ikLength.reso_zoom) \times \exp\left(i\frac{\omega_m}{c}2Length_cav\right) && \text{for the upper sidebands} \\
&\exp(ikLength.reso_zoom) \times \exp\left(-i\frac{\omega_m}{c}2Length_cav\right) && \text{for the lower sidebands}
\end{aligned} \tag{3.7}$$

As a reminder *Length_cav* is the length of the cavity and *Length.reso_zoom* is the microscopic tuning used to set the resonance condition for the carrier.

Since we know how to calculate the carrier and the sideband fields circulating in the cavity, we can also calculate the reflected fields from the cavity. The reflected light is also the superposition of three fields, the reflected carrier, the reflected lower sidebands and the reflected upper sidebands with respective amplitude E_0^{ref}, E_-^{ref} and E_+^{ref}. The power P_{det} detected by a photodiode placed in reflection will then be :

$$\begin{aligned} P_{det} &= |E_0^{ref} + E_-^{ref} e^{-i\omega_m t} + E_+^{ref} e^{i\omega_m t}|^2 \\ &= |E_0^{ref}|^2 + |E_-^{ref}|^2 + |E_+^{ref}|^2 \\ &\quad + \cos(\omega_m t)\Re(E_0^{ref} E_-^{ref*} + E_0^{ref*} E_+^{ref}) \\ &\quad + \sin(\omega_m t)\Im(E_0^{ref} E_-^{ref*} + E_0^{ref*} E_+^{ref}) \\ &\quad + 2\Re(E_+^{ref*} E_-^{ref} e^{-i2\omega_m t}) \end{aligned} \tag{3.8}$$

With $\Re$ and $\Im$ the real and imaginary part and $*$ designates the complex conjugate. The PDH error signal e_{PDH} is derived by demodulating in phase P_{det} and the resulting signal is then low-pass filtered. As a result e_{PDH} can simply be written:

$$e_{PDH} = \Im(E_0^{ref} E_-^{ref*} + E_0^{ref*} E_+^{ref}) \tag{3.9}$$

The Matlab implementation of the method described above is straightforward. We first define three electric fields: the carrier and then the lower and upper sidebands. Then we propagate each field in the cavity, calculate the reflected field and then plot the error signal using the formula 3.9. To make the result more pertinent, we fact scan the cavity over one free spectral range, so we can see the evolution of the error signal across the resonance.

To define the sideband frequency and amplitude, we have to define new variables (with self-explaining names) as shown in the script 3.3.

Then we can calculate the three circulating fields in the cavity called *Field.Total Field.Total_sidebands_plus* and *Field.Total_sidebands_minus* using the usual method introduced in section 2.4. The only difference in the algorithm between the propagation of the carrier and the sidebands is the addition of an extra phase shift per round-trip for the sidebands as shown in equation 3.7.

Listing 3.3: New variables to define the sidebands frequency and amplitude

```
% Frequency of the sidebands in Hertz and modulation index
Sidebands.frequency = 10E6;
Sidebands.modulation = 0.4;

% New carrier amplitude
Laser.amplitude = Laser.amplitude * besselj(0,Sidebands.modulation);
% Amplitude for one of the 2 sidebands
Sidebands.amplitude = besselj(1,Sidebands.modulation);
Ration_amp = Sidebands.amplitude/Laser.amplitude;

% To add the additional phase shift undergone by the sidebands during the propagation
Sidebands.k_prop = 2*pi*Sidebands.frequency/3E8;
```

Listing 3.4: Calculating the PDH error signals from the fields reflected by the cavity

```
Field.PDH = imag(Field.Reflect_carrier.*conj(Field.Reflect_sidebands_plus )+...
            conj(Field. Reflect_carrier ).*Field.Reflect_sidebands_minus);
% Normalised PDH signal
Power.PDH(r) = sum(sum(Field.PDH))/(Laser.amplitude*Sidebands.amplitude);
```

After calculating the three reflected fields from the cavity named *Field.Reflect_carrier* *Field.Reflect_sidebands_plus* and *Field.Reflect_sidebands_minus*, we can derive the PDH error signal using the formula shown by equation 3.9, the equivalent Matlab listing is shown in 3.4.

The plot of the power circulating in the cavity as a function of the microscopic de-tuning is presented in figure 3.7. The central peak is due to the resonance of the carrier and the two smaller peaks indicates the resonance position for the sidebands. Finally the usual PDH error signal as calculated by OSCAR is shown in figure 3.8. Around the resonance of the carrier, we have an error signal to lock the cavity. In theory, we have also an error signal to lock the cavity on the sideband resonance (less useful).

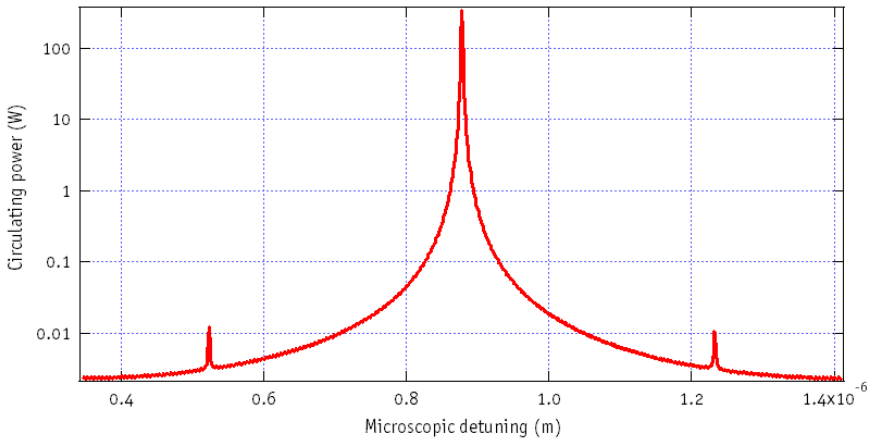


Figure 3.7: Circulating power inside the cavity as the cavity is scanned over one FSR. The two small peaks around the main resonance indicate the resonance of the sidebands.

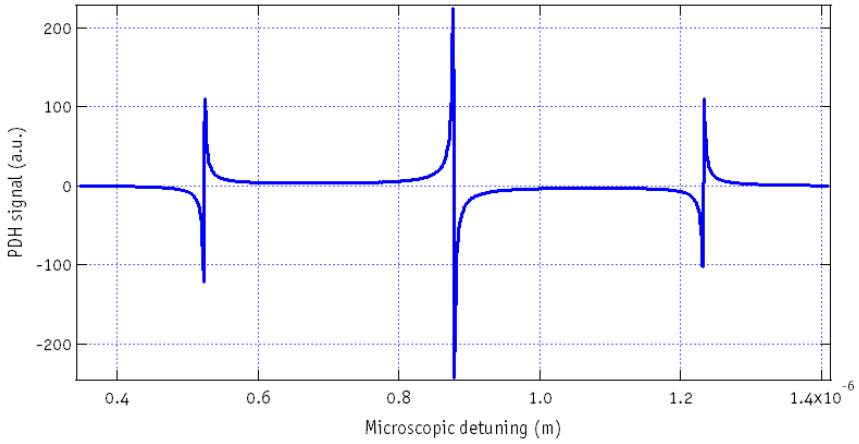


Figure 3.8: Pound-Drever-Hall error signal as a function of the de-tuning.

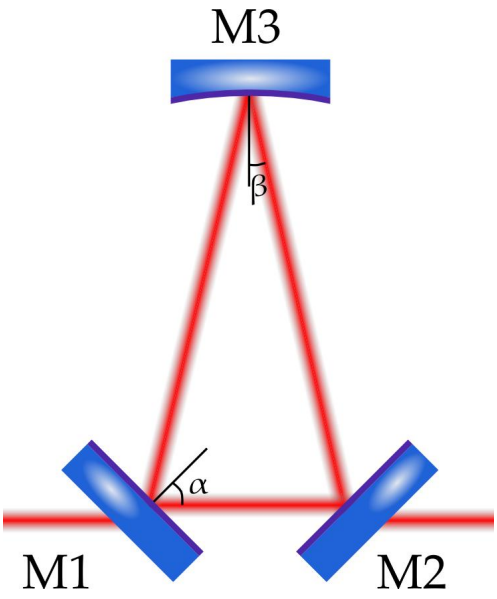


Figure 3.9: Optical setup of a three-mirror ring cavity. Unlike a Fabry-Perot cavity, where the mirrors are usually perpendicular to the incident beam, for the mode-cleaner the incoming beam arrives with a large angle on the two bottom mirrors. The mirror notation used in the OSCAR example are also indicated.

3.5 Simulation of a three-mirror ring cavity

We can try now to add an additional mirror to our simulations and simulate a three mirrors ring cavity, often used as a mode-cleaner. In the previous example we have been dealing with an optical beam perpendicular to the optical surface, this is no longer the case as seen in figure 3.9. The fact that the incident beam arrives with an angle to the surface is the main difficultly for this kind of simulation. However as we will see this problem can easily be overcome.

In the OSCAR mode-cleaner example (folder Simulate_mode_cleaner), we suppose that the incident plane is horizontal. We keep this convention in the manual and it is relatively easy to adapt the code for any arbitrary angle of incidence. Let's examine what is happening on reflection: as soon as the angle of incidence of an electric field to a reflective surface is different from zero, two effects can arise:

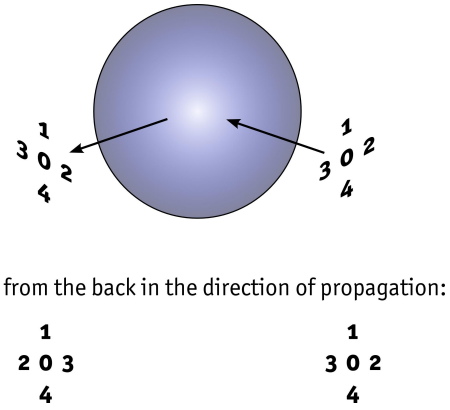


Figure 3.10: Example of an image reflected by a mirror. It is interesting to see that after reflection the image is flipped in the horizontal direction.

- The reflected beam will present astigmatism directly related to the angle of incidence. That can be understood by imagining that the incident beam will be stretched in the horizontal direction as it is projected to the mirror surface upon reflection.
- The beam on reflection will get flipped in the horizontal direction, this is purely a geometrical effect and the reader can easily be convinced by looking at the figure 3.10.

The two items described above are implemented in OSCAR in the new function *Propa_mirror* shown in the listing 3.5. Compared to the function *Propa_mirror* seen in the first chapter (section 1.3.5), we have added one new input parameter: *Grid_angle_X*. *Grid_angle_X* is a new distorted 2D grid representing the projection of the grid used for propagation (called *Grid.D2*) to the mirror surface.

The function used to simulate the reflection on a mirror can be decomposed in the following steps:

1. Project the incident beam on the mirror surface, so that the incident beam will look astigmatic
2. Reflect the projected incident beam in the same way as a incoming beam normal to the mirror surface
3. Project the reflected beam on the plane of propagation

Listing 3.5: Reflection on a mirror for arbitrary angle of incidence

```
% Stretch the laser beam as seen by the mirror
Output = Propa_mirror(Wave_field, Wave_mirror, reflec,Grid_angle_X)

Output = interp2(Grid_angle_X,Grid.Y,Wave_field,Grid.X,Grid.Y,'cubic');

Output = Output .* exp(-i * Wave_mirror*Laser.k_prop) * reflec .* Mirror.mask;


% Go back to the normal grid
Output = interp2(Grid.X,Grid.Y,Output,Grid_angle_X,Grid.Y,'cubic',0);

% Flip the matrix along the x axis
Output = Output(:,Grid.Num_point:-1:1);
```

4. Flip the beam in the horizontal plane

The steps described above are directly implemented in Matlab as shown in the script 3.5.

In the examples provided with OSCAR, the transmission of a three-mirror ring cavity is presented. To show the mode-cleaning effect, the input beam is the normalized sum of the TEM_{10} and TEM_{01}. As expected both modes have different resonance condition inside the mode-cleaner as shown in figure 3.11. The microscopic resonance length of the two modes is separated by half a wavelength, which is the expected result since the mode TEM_{10} encountered an extra 3π shift during its round-trip propagation.

For this example, we set the working point of the cavity for the mode TEM_{10} to be resonant inside the cavity. The transmitted field from the mode-cleaner is shown at the bottom right corner of figure 3.12. As we would hope, the transmitted field is a pure TEM_{10}. For consistency, we also check that the reflected field is very similar to a pure TEM_{01}, but not exactly since the input beam was not perfectly matched to the cavity.

Below is the output from OSCAR:

```
------ Display the results ------
Circulating power (W): 49.524735
Reflected power (W): 0.503506
Transmitted power (W): 0.495243
```

So the mode-cleaner was able to separate the mode TEM_{10} and TEM_{01}. We have slightly more power reflected than transmitted by the cavity, this is

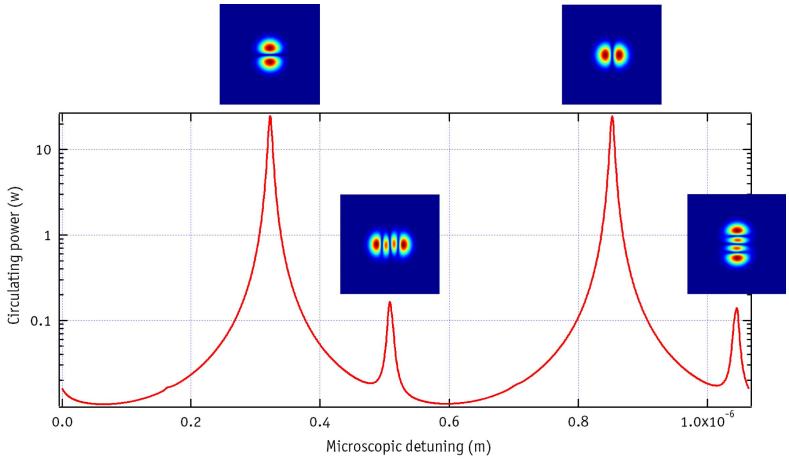


Figure 3.11: Scan of the mode-cleaner over one free spectral range. The two main peaks are due to the resonance of the TEM_{01} and TEM_{10}. Additional peaks in the spectrum indicates a slight mode-mismatching.

a direct consequence of the fact that the input beam is not perfectly matched to the cavity.

```
---- Display results for cavity C1 -----
Round trip diffraction loss: 35.6127 [ppm]
Circulating power: 49.8224 [W]
Size of the beam on the end mirror: 0.0205821 [m]
Size of the beam on the input mirror: 0.020576 [m]
Size of the cavity waist: 0.0183982 [m]
Distance of the cavity waist from the input mirror: -500.471 [m]
```

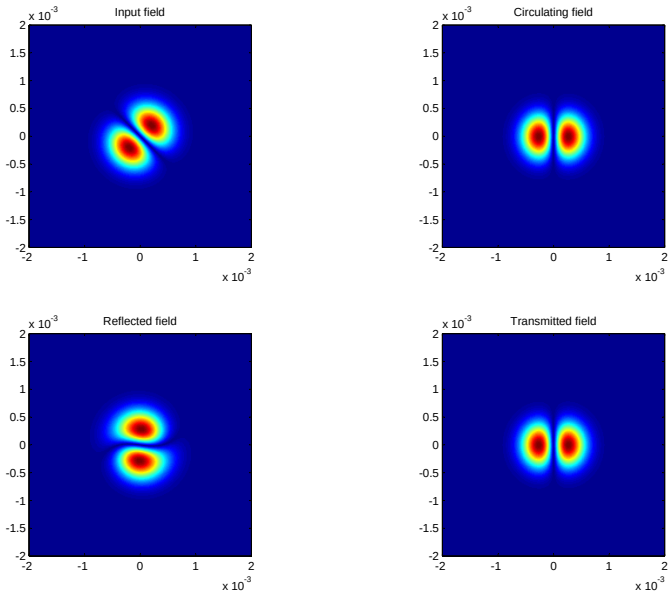


Figure 3.12: Output screen from OSCAR. The input field, a sum of TEM_{10} and TEM_{01} is shown on the top left corner. The reflected and the transmitted field are shown respectively in the bottom left and right picture.

Part II

OSCAR V3: the Object-Oriented version

Chapter 4

The big changes

Since the version 3.0, the code has been totally rewritten using the Object-Oriented capability of Matlab. The code is now user-friendlier and less error-prone. Moreover, some new functions have been added and the higher order optical modes are supported from the start.

The principle of the FFT code and the method to derive the results are still unchanged, so for people not familiar with the inner working of FFT code, it is strongly recommended to read the first part of this manual.

Requirements The new OSCAR works with Matlab version superior to 2009a and the *Optimization Toolbox* is necessary if the beam fitting functions are used. To benefit from the GPU speed up, the *Parallel Computing Toolbox* is also required.

4.1 Motivation and goals

First, the curious reader may wonder why there is no version 2.0 of OSCAR. Such a version exists, it is OSCAR classic with the implementation of a power recycled Michelson with Fabry-Perot arm cavities. This version was developed to simulate LIGO-Virgo-type interferometers but was never officially distributed.

The code of version 2.0 was becoming messy since OSCAR was not intended to be so advanced (remember, it was only supposed to simulate the test cavity at Gingin in Australia). To include all the code developed for the version 2.0, it was decided to write a full new version of OSCAR taking also into account everything I learned as a OSCAR user.

Writing a new version allowed some radical changes: no more global variables, possible change of refractive index, sideband field managed automatically at the same time as the carrier, higher order modes and much more. In the long term, OSCAR should be able to simulate Advanced gravitational wave detectors with realistic optics, and that in a (relatively) very simple script.

The new code should also take into account the computing progress made in recent years. For example, the version 3.0 has been written with parallel processing in mind, for example the North and West arms could be computed in parallel on different cores of the CPU (the Parallel Processing Toolbox is then required) or a cavity scan can take advantage of multicore CPU. Since version 3.30, calculations on the GPU are enabled.

The new code can also use separate grid sizes in the same simulation. That could be used to simulate non-degenerate recycling cavities, where the laser beam has to be strongly focused and could change size by a factor 10.

4.2 The new objects and how to use them

The main change of this version for the user is the implementation of Object-Oriented programming for OSCAR. A laser field or a mirror are no longer complex 2D matrices but are objects with different attributes. Each object is part of a class, for example the laser beam is an instance of the class *E_field*, a mirror surface is an object from the class *Interface*. Functions are defined to work on specific classes and not on any random 2D data.

The definition of the classes and their associated functions are presented in the following subsections. In the description of the function call, argument and output variables in **bold** are mandatory. For more detailed descriptions of the functions and the associated syntax, the help of Matlab can be used (type "help" followed by the name of the function).

4.2.1 The class Grid

G1 = Grid(Nb_points_grid,Size_grid) by default: Nb_points_grid = 256 and Size_grid = 1

The class *Grid* is used to represent the discretization of space and all the associated variables (see C.1 for the details). It was formerly known as the global variable *Grid* in OSCAR classic. One *Grid* object is the first object to be defined in a OSCAR script. To define an instance of the class *Grid*, 2

parameters are necessary: the number of pixels used (typically 128 or 256) and the size of the grid in meters. For example:

```
G1 = Grid(128,0.40);
```

Defines the object *G1* of the class *Grid*, then later all the objects for the simulations will be defined on a grid of 128×128 with a side length of 40 cm.

Only two functions are associated with the class *Grid* : What_is_possible and Do_Virtual_Map.

What_Is_Possible(**G1**) returns for the grid *G1* what is the approximate maximum curvature or tilt for a mirror. By default the mirror diameter is 80% of the size of the grid and the wavelength is set to 1064 nm.

It is possible to change manually the diameter of the mirror and the wavelength of the laser by using the optional arguments *'Diam'* and *'Wavelength'*:

```
What_is_possible(G1,'Diam',0.2,'Wavelength',633E-9)
```

map2D = Do_Virtual_Map(**G**, **Power_law**)

> Creates a synthetic map from a given PSD. The PSD is parameterized as a (or several) power law. The function return a 2D height map of the surface which can then after be applied to a Interface object. See the example Example_Display_Create_maps.m for a practical implementation.

4.2.2 The class E_field

E1 = E_field(**Grid_name** , *options*) by default the laser beam is in the fundamental mode.

Instances of the class *E_field* are laser beams, represented as electric fields in a complex 2D matrix. The objects *E_field* contain a wavelength (1064 nm by default) and also the refractive index of the medium where the laser beam is. If sidebands are present, they are also implemented in the object at the same time as the carrier. All the variables and parameters of objects *E_field* are detailed in C.2.

Since OSCAR version 3.05, there are now different ways to define a laser beam:

- with the waist:
  ```
  E_input = E_Field(G1,'w0',0.02)
  ```
- with the waist and a distance:
  ```
  E_input = E_Field(G1,'w0',0.02,'z',-10)
  ```

- with a beam radius and a wavefront curvature:
  `E_input = E_Field(G1,'w',0.018,'R',-1420)`
- with the complex q parameter:
  `E_input = E_Field(G1,'q',12 + 1200*1i)`
- the optical mode can also be specified, for example for a helicoidal mode LG 3 3: `E_input = E_Field(G1,'w0',0.02,'mode','LG_HELI 3 3')` To define the mode basis, 3 options are possible: 'HG n m', 'LG_HELI n m' or 'LG_SIN n m'
- one can also specify the wavelength (in meters):
  `E_input = E_Field(G1,'w0',0.02,'Wavelength',532E-9)`

A non distributed function also exists to create astigmatic beams. Most of OSCAR's functions are applied to objects of this class. The essential functions are described below:

E2 = Add_HOM(**E1**, **max_mode_order**)

Adds some higher order optical modes to a fundamental mode. It is possible to choose the power distribution of the higher order modes.

E2 = Add_Sidebands(**E1**,'Mod_freq',**mod freq**,'Mod_depth',**mod index**)

Creates two frequency-shifted sideband fields from a carrier E_Field *E1* as the phase modulator would do in the optic lab. The separation frequency between the carrier and the sideband is given by *mod freq*, and the modulation index by *mod index*. At present in OSCAR, no sidebands of sidebands are allowed but an arbitrary number of sidebands can be created, since the version 3.30.

E2 = Add_Tilt(**E1**,**angle_tilt**)

Tilts the wavefront of the field **E1** by the angle **angle_tilt** given in radian. The maximum tilt allowed depends on the pixel size but it is usually a small number (order of the milliradian or less).

overlap = Calculate_Overlap(**E1,E2**)

Calculates the overlap integral between 2 E_field. The function returns the complex normalized overlap between the 2 fields. Without any output argument, the function returns in the command window the power coupling (i.e. square of the absolute value of

the overlap integral) between the field E1 and E2. If only one E_field is given, it will calculate the overlap between the carrier and of the sideband pair. The sideband number is selected with the argument 'SB_num'.

power = Calculate_Power(**E1**,'include','carrier'—'all'—'SB')

Calculates the total power of the E_Field *E1*. The function can either return a number or display the value in the Matlab command line if no output variable is used. With the same function, one can also calculate the power of the sideband fields: Calculate_power(E1,'include','SB','SB_num',1)

E2 = Change_E_n(**E1**,**n**)

Changes the refractive index where the laser beam is. Could be used to pass through a flat interface from a vacuum to a mirror substrate for example.

return_param = Check_Pos_Tilt(**E1**)

Uses as a diagnostic tool to check the centroid position of the beam on the grid as well as its tilt, function in both directions at the same time.

[**sig_p sig_q**] = Demodulate_SB(**E1**,'SB_num',1,'phase',0)

Demodulates the field *E1* with the chosen pair of sidebands. By default, the in phase *sig_p* and in quadrature *sig_q* signals are returned. This function could be used to derive a Pound-Drever-Hall error signal as shown in 5.5. Optionally, a demodulation phase could be given.

E_Plot(**E1**)

Displays a 2D plot of the field amplitude in a new Matlab figure window. Function can also be used as *E1*.E_plot(). We can also display the amplitude in log scale `E_plot(E1,'space','log')` or the propagation angle of the beam `E_plot(E1,'angle')`. To create illustrative picture of the beam without all the axis, one can use the option `'no_axis',true`.

The same function could be used to display the 2 sidebands with E_Plot(E1,'display','SB','SB_num',1). The same options as for the carrier are also possible.

Mat_scattered = Expand_HOM(**E1**,**max_mode_order**)

Expands the beam *E1* into the Hermite-Gauss basis defined by *E1* (*E1* must be close to a fundamental mode). The result is given as the normalized power content in the different optical modes. If not output is asked a plot is made. Several options are possible:

- output all the power content in the different higher order modes up to the order 10
  ```
  Expand_HOM(E1,10,'display','pyramid')
  ```
- define a different basis for the calculation given as beam radius and complex wavefront curvature in meters
  ```
  Expand_HOM(E1,10,'basis',[0.05 -1400])
  ```

[beam_radius wavefront_RofC] = Fit_E_Field(**E1**)

This function is used to find the beam parameter of the field *E1*, it can also be used to fit higher order modes. If no output are given the result is written in the Matlab command window. To fit only the fundamental mode, it is faster to use the function Fit_TEM00.

For the fit of an astigmatic fundamental mode, it is recommended to use the function Fit_TEM00_xy.

[**Eout**,**Gout**] = Focus_Beam_With_Telescope(**Ein**,**array_L_D**)

With this function, one can now simulate mirrors with small radius of curvature to simulate strongly focused beam. The method has been inspired by the work done J.Y. Vinet [20] and H. Yamamoto [21]. This function can dynamically adapt the size of the grid and so the focused beam could be defined on a new grid. This function is used to simulate the focusing mode-matching telescope at the input or output of the interferometer. The input argument is the input field and a vector with the focal length of the equivalent mirror/lens and the separation between the optical elements in meters. A telescope with several lenses can be simulated with only one call to the function. See the provided example for more details. It has also the possibility to add maps and finite aperture of the optics of the telescope.

E2 = Normalise_E(**E1**,power)

Normalizes the E_field *E1* to 1 W if no *power* argument is specified. Only the carrier is affected by this function and not the sidebands.

E2 = Propagate_E(**E1**,**dist**)

Propagates the field *E1* over a distance *dist* in meters. If sidebands are present, the sidebands are automatically propagated at the same time, with of course the adequate phase shift.

E2 = Reflect_Mirror(**E1**,**I1**)

Returns the field after reflection on a mirror. *I1* is an Interface object which must have been previously defined. An alternative syntax could be Reflect_mirror(**E1**,**R1**), with *R1* the radius of curvature of the mirror to be reflected on. A concave mirror is represented by a positive **R1**.

Reflections inside the substrates are allowed since the index of refraction is taken into account for the reflection.

E3 = Remove_00_Part(**E1**,E2)

Removes the fundamental Gaussian part from a field **E1**. That is useful to see which higher order modes are superposed with the fundamental mode following some cavities simulations. Can also be used to subtract 2 fields E1 and E2 as in the case of destructive interference.

E2 = Transmit_Aperture(**E1**,**size**, 'circle' | 'square')

Transmits the field *E1* through an aperture of diameter *size*. The aperture could be round or square, if not specified it is round. Custom aperture can also be defined (see the function help for examples).

E2 = Transmit_Lens(**E1**,**focal length**)

Transmits the field *E1* through a lens of a given focal length.

[**E_trans** E_ref]= Transmit_Reflect_Interface(**E1**,**I1**)

Transmits and reflects the field *E1* through an interface *I1*. The orientation of the interface is given by the refractive index of the incoming field *E1*.

[**E_trans** E_ref]= Transmit_Reflect_Mirror(**E1**,**M1**,**HR/AR**)

Transmits and reflects the field *E1* through a mirror *M1*. One must select which side of the mirror is first met for the propagation, either we enter by the AR or HR side.

For convenience, the operators +,- and * have also been overloaded to work with any object *E_field*.

4.2.3 The class Prop_operator

This is an internal class which must be almost invisible for the user. The objects of this class allow to speed up the propagation of electrical field *E_Field* thanks to the pre-calculation of the propagation matrix. A *Prop_operator* is automatically created when a cavity is created. It represents the propagation operator over one cavity length.

Since OSCAR 3.10, a new method of propagation has been implemented based on the method of digital integration [22]. This method is slower, however it eliminates the light leaking out from the simulation grid. With the direct implementation, a minimal distance of propagation is essential for the validity of this technique. The distance depends on the size of the grid and can be checked with the function What_is_possible.

To enable the digital integration, one has to set the logical variable *Prop_operator.Use_DI* to true. For example, to use the digital integration for the simulation of the cavity *C1*, we will write:

```
C1.Propagation_mat.Use_DI = true;
```

The parameters included in the class *Prop_operator* are described in C.3

The creator for this class is:

Prop = Prop_operator(**Grid_name**,**Length**, n)

> The variable *Length* represents the propagation length in meters. One can also specify in an optional parameter *n* the refractive index of the media, so that thick substrate can also be implemented.

To propagate a laser beam (and its sidebands) one can simply use:

```
E2 = Propagate_E(E1, Prop)
```

4.2.4 The class Interface

I1 = Interface(**Grid_name**,options)

The class *Interface* is used to represent a surface which separates two media of different refractive index. Objects of this class are used to represent mirrors, lenses or curved interfaces between air and glass. The structure of the class *Interface* is described in C.5.

To define a concave mirror for a laser beam, the radius of curvature must be positive. Internally in OSCAR, the sagitta of the surface viewed from the medium $n1$ toward $n2$ will be negative by convention. The surface is stored in the variable *I1.surface*.

Below are some ways to define an interface:

- Interface to define a curved surface of radius 1000 m between the refractive indices 1 and 1.45
  `Interface(Grid,'RoC',1E3,'n1',1,'n2',1.45)`
- Mirror with 20 cm diameter aperture
  `Interface(Grid,'RoC',1E3,'CA',0.2)`
- Mirror with 20 cm diameter aperture and 3 degrees angle of incidence
  `Interface(Grid,'RoC',1E3,'CA',0.2,'AoI',3)`
- Add a 1% transmission and 50 ppm loss
  `Interface(Grid,'RoC',1E3,'CA',0.2.'T',0.01,'L',50E-6)`

The optional parameters can be defined in any order. To define flat mirrors, a `'RoC'` of `0 or Inf` can be used. Check the file `@Interface/Interface.m` to see the default parameters such as refractive indexes.

The list of methods associated with the objects of the class *Interface* is given below:

I2 = Add_Astigmatism(**I1**, **Zernike amplitude**, **Diameter**)

Adds to the surface of the interface *I1* the Zernike polynomial 2,2 (which represents the astigmatism) with an amplitude of *Zernike amplitude* and over a diameter *Diameter*.

I2 = Add_Map(**I1**, **filename**, options)

Adds to the surface of the interface *I1* a square map loaded from the file *filename*. For square maps, the resolution of one pixel must be given by the parameters `'reso'`. The map can also be normalized to a certain RMS using the option `'RMS'` or the height can be scaled by scalar with the option `'scale'`. One can also rotate the map by a number of times 90 degrees with the option `'rotate'`. The option `'remove the tilt and focus'` can remove the tilt and focus from the added map. The tilt/focus is removed over all the surface but with the optimal fit calculated only over a diameter.

Here some examples:

- Load the file map1.dat
  `I1 = Add_map(I1,'map1.dat','reso',3.5E-4)`
- Load the map and scale it to 1 nm RMS
  `I1 = Add_map(I1,'map1.dat','reso',3.5E-4,'RMS',1E-9)`

- Rotate the map by 90 degree
  `I1 = Add_map(I1,'map1.dat','reso',3.5E-4,'rotate',1)`
- Remove the tilt and focus
  `I1 = Add_map(I1,'map1.dat','reso',3.5E-4,'remove_tilt_focus',0.150)`

Maps with cylindrical symmetry can be loaded as a 2-column text files. First column is the radius and the second is the height in meters. In that case the resolution does not have to be specified. To suppress the output, the option `'verbose',false` can be used. A new argument is possible to offset the map `'shift'` if the region of interest is not at the center of the map.

I2 = Add_Tilt(**I1**, **tilt_angle**, 'x')

Tilts the whole surface of the interface *I1* by the amount *tilt_angle* in radian. It is possible to specify the direction of the tilt by setting the third argument *'x'* or *'y'*. By default the tilt is in the vertical direction (recommended solution).

I2 = Cut_Frequency_Interface(**I1**,**Option**,**f_cut**)

Removes certain spatial frequencies within a given interface object. The argument *Option* is a string `'LP'`, `'HP'` or `'BP'` for respectively low pass, high pass or band pass filter. *f_cut* is the corner spatial frequency in m^{-1}, for band pass filtering *f_cut* is vector of two frequencies.
The implementation for the filtering is done in a straightforward manner (see the simple code), so that artifacts may appear in the resulting object *I2* and that this function should be used with caution.

I2 = Expand_Zernike(**I1**,options)

Expands in Zernike polynomials the interface *I1*. Two options are available: 'Z_order', an integer which represents the maximum order of the Zernike polynomials to take into account in the calculations and 'diam', the diameter where the Zernike polynomials are defined. This function returns the interface *I2*, which is the interface reconstructed by the Zernike polynomials. Example:
`I2 = Expand_Zernike(I1,'Z_order',10,'diam',0.2)`

I_Plot(**I1**)

Plots the surface of the interface *I1*. The color scale is in meters. `I_plot(I1,'diam',0.160)` will only plot the surface over a diameter of 160mm. With optional arguments, one can change the scale of the vertical axis and also zoom in the central part.

[**PSD_1D freq**] = Plot_PSD(**I1**, options)

Function used to calculate the one dimension Power Spectral Density (PSD) of a surface. The two output arguments are the vector of the PSD and the vector of the frequency. By default, before calculating the PSD 2D, a Hanning window is applied to the data and the PSD is calculated summing all the frequencies over one direction to pass from 2D to 1D. A detailed explanation of the different ways to calculate the PSD can be found in a Hiro Yamamoto's LIGO note [23].

An example for the possible options can be found below, combination of the different options is allowed:

- Calculate the PSD over a given diameter, given in meters
  `[PSD_1 freq] = Plot_PSD(I1,'diam',0.150)`
- Calculate the PSD by summing radial frequencies
  `[PSD_1 freq] = Plot_PSD(I1,'rect_1D',false)`
- Use a Gaussian weighting to calculate the PSD, must input the Gaussian beam radius (in meters)
  `[PSD_1 freq] = Plot_PSD(I1,'window_Gaussian',0.05)`
- Plot directly the PSD in the current window
  `[PSD_1 freq] = Plot_PSD(I1,'display',true)`

RMS = Weighted_RMS(**I1**,'E',**Ein**)

Calculate the RMS of the surface *I1* weighted by the intensity of the field *Ein*. A diameter can also be given as the second argument. Beware that the curvature is not subtracted before the calculation, so that a perfectly curved spherical mirror will have a non zero RMS.

- Calculate the weighted RoC
  `RMS = Weighted_RMS(I1,'E',E_Field(G1,'w',0.05))`
- Display directly the calculated RMS
  `Weighted_RMS(I1,'diam',0.2)`

RoC_fitted = Weighted_RoC(**I1**,'E',**Ein**)

Calculate the radius of curvature of the surface *I1* weighted by the intensity of the field *Ein*. This calculation is useful to fit non perfectly spherical surfaces. A diameter can also be given as the second argument.

- Calculate the weighted RoC
  `RoC = Weighted_RoC(I1,'E',E_Field(G1,'w',0.05))`
- Display directly the radius of curvature
  `Weighted_RoC(I1,'diam',0.2)`

The operators `+`,`-` have also been overloaded to be used with objects of the class *Interface*.

4.2.5 The class Mirror

M = Mirror(**I_HR**, **I_AR**, **Substrate_length**)

This class can simulate thick substrates. The substrate can be part of a cavity as an input mirror for example. If the substrate is part of the cavity, the surface *I_HR* is supposed to be the HR coating (and hence inside the cavity).

I_HR and *I_AR* are 2 objects of the class *Interface*. The 2 interfaces must share the same refractive index superior to 1. *Substrate_length* is a scalar used to represent the thickness of the substrate in meters.

The mirror object can also be used to simulate the etalon effect. Indeed if the AR surface is imperfect (R_AR $>$ 0), light can be bounced several times within the substrate before exiting. The number of round-trips is hard-coded but can be changed using the line (by default it is 1):

```
M.RT_inside=3
```

Two similar procedures can be used with the class *Mirror*. The procedures simulate the transmission and reflection of an E_field by an instance of the class *Mirror*:

[**E_trans** E_ref]= Transmit_Reflect_Mirror(**E1**,**M1**,'AR')

Calculates the transmitted and reflected fields (*E_trans* and *E_ref* respectively) by an object *M1* of the class Mirror. The input beam is *E1*. The first interface to consider, indicating the direction of the incoming beam can be either the HR or AR surfaces according to the last argument 'HR' or 'AR' .

A new wrapper function Transmit_Reflect_Optic has also been created which can be used by either Interface object or Mirror object. Internally, this function calls the function Transmit_Reflect_Interface or Transmit_Reflect_Mirror depending on the input parameters.

4.2.6 The class Cavity1

C1 = Cavity1(**I1**, **I2**, **Cavity_length**, **E1**)

This class is used to simulate Fabry Perot cavities. Objects of the class *Cavity1* are cavities constituted by two Interface objects *I1, I2* separated by a distance *Cavity_length*. An input field *E1* must also be given. *I1* is considered as the input surface where the input beam is defined, whereas *I2* is the end mirror.

The input field can be defined either inside the cavity on the surface *I1* or outside the cavity. In the later case, the input beam will first be transmitted through *I1* which can act as a lens. By default the input beam is defined inside the cavity, if it is not the case the user can add in his script:

```
C1.Laser_start_on_input = false ;
```

Variables of instances of the class *Cavity1* are listed in App3:Cavity. Several functions are associated with the class *Cavity1*:

Cout = Calculate_Fields(**Cin**)

Calculate_fields calculates the fields on resonance inside the cavity and stores them for future use. Compared to Get_info this function also calculates the reflected field, so the input laser beam cannot be defined inside the cavity. The results can be displayed using Display_results(Cout). Before finding the steady state fields inside the cavity, the cavity working point (i.e the resonance condition) has to be set, for example using the function Cavity_resonance_phase. With the option argument `'iter'` or `'accuracy'`, one can set the level of accuracy (and hence the computational time) for the simulation.

Cout = Calculate_Fields_AC(**Cin**)

Calculate_fields_AC is similar to the function Calculate_fields except that it uses the accelerated convergence scheme [12, 24] to find the steady state fields inside the cavity. The gain in speed could be important if the cavity is close to a perfect cavity, so the steady field is near a perfect fundamental mode. This function works for the carrier and for the sideband fields, if present. For a better convergence, it is also recommended to use the digital integration method for the propagation of the field (see 4.2.3). A second function Calculate_fields_AC2 uses another more advanced convergence scheme which can give more accurate results.

Cout = Calculate_RT_mat(**Cin**)

Calculates the round-trip kernel of the cavity for the electrical field. This is later use to calculate the cavity eigenmodes and eigenvalues. The kernel itself is a 2D complex matrix of size $Nb\_points^2 \times Nb\_points^2$, so the kernel can really occupy a lot of memory. On a desktop, one might want to limit *Nb_points* to 64, while on a powerful desktop one can go up to 128.

To calculate the kernel, the digital integration technique has to be used for the beam propagation. To be valid, this technique requires a minimum propagation distance which depends on the size of the grid. If this is a problem, the kernel can be calculated on a different grid (contact the author for more details on this topic).

Cout = Cavity_lock_PDH(**Cin**)

Adjusts the resonance condition to minimize the error signal derived from a Pound-Drever-Hall locking scheme [19]. To calculate the error signal a pair of sidebands must be present and the function Cavity_resonance_phase must first be run.

Check_stability(**Cin**)

Calculates the radius of curvature of the input and end mirrors and then check the stability and gain of the cavity. It is a useful function when mirror maps with curvature are used. An optional second argument could be given to specify the diameter on which the mirror radius of curvature is calculated. For example to calculate the stability of the cavity only taking into account the central 6 cm diameter of the mirror, one can use: `Check_stability(C1,'diam',0.06)` The function also returns the input beam parameters of the fundamental mode for a perfect mode-matching.

Check_Matching(**Cin**,nb_RT)

Propagates the input beam back and forth between the cavity input and end mirrors and checks the beam size before each reflection. The mode-matching is good if during multiple round-trips the beam size on the mirror does not fluctuate (but could be different of course on the input and end mirror). The optional argument *nb_RT* can set the number of round-trips to be done.

Cout = Cavity_Resonance_Phase(**Cin**)

Calculates the macroscopic length tuning to bring the cavity on resonance for the input field. The cavity resonance has to be found before any calculations on the cavity (circulating fields, loss calculations,...) are launched. The input and output arguments are usually the same. A second function Cavity_resonance_phase2 is also present, this function gives more accurate results when the cavity is degenerate.

Cout = Cavity_Scan(**Cin**)

Scans the cavity over one free spectral range and then sets the cavity resonance to maximize the circulating power. This procedure is much slower than Cavity_Resonance_Phase but can still be useful to check if some higher order modes are excited, since afterwards the FSR scan can be displayed. The presence of sidebands in the input beam of the cavity can also be implemented. First the sidebands have to be defined and then the option `'With_SB',true` must be added in the options.

Cout = Declare_on_GPU(**Cin**)

Load all the pertinent variables regarding the cavity on the GPU. In that way, the most intensive calculations will be done on the GPU with some potential gain in calculation time, especially for large grids.

The most gain in time is for the cavity FSR scans and the implementation is detailed in a dedicated paper [25].

Display_Cavity_Modes(**Cin**,'N',10,'Airy',1)

Displays the cavity eigenmodes. The function Calculate_RT_mat must be run first to calculate the cavity kernel. With this function one can display an arbitrary number of modes (by default the first 20) and also draw the Airy peak of the cavity with the command:

```
Display\_cavity\_modes(C1,'N',30,'Airy',1)
```

An example of how to use the function Display_cavity_modes is shown in the script Example_cavity_eigenmodes.m

Display_Scan(**Cin**)

Displays the scan from a cavity after the procedure Cavity_scan was run. By moving the cursor on the upper plot (circulating power vs tuning), the profile of the optical modes can be checked in the lower right plot.

Get_Info(**Cin**)

Get_info calculates the fields on resonance inside the cavity. The function is also used to calculate the diffraction loss due to the mirror finite size. Legacy function, it is recommended to use the function Calculate_Fields_AC.

4.2.7 The class CavityN

CN = CavityN(**I**, **D**, **E1**)

This class introduces linear cavities with arbitrary number of mirrors. *I* represents a vector of object Interfaces, *D* is a vector of distances and finally *E1* is the input laser field.

Two kinds of cavity could be defined:

- A ring cavity where the laser beam is reflected on each mirror only once during one round-trip (3- or 4-mirror mode-cleaner cavities fall in this category). In that case, the length of the vector *I* is the same as the one of *D*.
- A folded cavity, where during one round-trip the laser beam is reflected twice by the intermediate mirrors (think recycling folded cavity of LIGO). In that case, it has one more interface *I* than length *D*.

The distance `D(n)` is defined as the distance between the interfaces `I(n)` and `I(n+1)`. The functions suitable for the class *CavityN* are the same as for the class *Cavity1* without exception.

An example of how to use the class *CavityN* is shown in the script Example_4mirrors.m

4.2.8 The class Mirror

This class is used to simulate optics with thick substrates. A mirror is thus defined as 2 surfaces separated by a distance with a certain refractive index. Typically, one surface is coated with an anti-reflective coating while the second surface has a high reflectivity coating.

Similar to a cavity, the light can do several round-trips within the substrate if the anti-reflection is not perfect. So the Mirror object can be used to simulate the etalon effect within a substrate. It is not possible to set the resonance condition in the substrate but one can scan the substrate length over wavelength to check this effect.

Mirror object can be used with the class *Cavity1* to form a cavity between the 2 high reflectivity surfaces.

4.3 What to expect in the following versions

Most of the code for the release V3.0 was frozen in September 2012. Some additional functions have already been implemented and are little by little moving in the official distribution. The motor behind most the new development is the need to simulate accurately the off axis secondary beams in the Advanced Virgo marginally stable recycling cavities.

Two main new features have already been implemented but not released:

- **Use of rectangular grid.** It is possible to use grid with different size and number of points in the horizontal and vertical directions. That is useful to simulate tilted beams with large angle along one direction since in that case a fine resolution is required. A special version of OSCAR in 1D has also been made but may never get released.
- **Simulation of a dual recycled Michelson interferometer.** A new class has been defined to simulate a dual recycled Michelson interferometer. That is done for marginally stable cavity but not yet released due to a lack of comprehensive documentation. Available for people within the Virgo collaboration.
- **Simulation birefringence.**. To include 2D maps of birefringence measured in transmission.

If new functions are needed urgently and cannot wait for future versions, you can contact directly the author.

Chapter 5

New examples of simulations

After the formal previous chapter, here is the most interesting part: the concrete examples. In this part we will see how easy it is to create simulations with the new OSCAR, few lines of code are usually enough to get a result. The examples are in the folder *Examples*.

5.1 Beam parameters after a thick lens

Before learning how to define cavities, we can already see how to propagate a laser beam through a custom-made thick lens. The lens is bi-convex, made of fused silica (n=1.45) and is 4 cm thick . The first surface has a radius of 1 km and the second 2 km. At the exit of the lens, we propagate the beam further by 200 m and then measure its parameter. The commented script is called Example_thick_lens.m and can be found in the OSCAR folder.

The first surface is defined in OSCAR by the line:
`L1 = Interface(G1,-1000,0.10,1,0,1,1.45)`. Since the surface is convex as seen from the incoming laser beam there is a minus sign. The second argument is the clear aperture of the optic, then there are the transmission and the loss (in power) and finally the two refractive index of the media on either part of the interface.

To transmit the laser beam `E1` through the interface, the command is simply `E2 = Transmit_Reflect_Interface(E1,L1)`. Then the beam is propagated along 4 cm with the command `E2 = Propagate_E(E2,0.04)` and finally passes through the second interface.

At the end of the script, the laser beam parameter is displayed using the command `Fit_TEM00(E3)`. The condensed script for the transmission of the

thick lens is shown in the listing 5.1.

Listing 5.1: Example of OSCAR script to simulate a thick lens

```
G1 = Grid(256,0.3);
E1 = E_Field(G1,0.02);

L1 = Interface(G1,-1000,0.10,1,0,1,1.45);
L2 = Interface(G1 ,2000,0.10,1,0,1.45,1);

E2 = Transmit_Reflect_Interface(E1,L1);
E2 = Propagate_E(E2,0.04);
E2 = Transmit_Reflect_Interface(E2,L2);

E3 = Propagate_E(E2,200);
Fit_TEM00(E3);
```

At the end, the beam radius is measured to be 1.76283 cm and the wavefront curvature -1765.36 m. To be compared with the result found using the ABCD matrix for the same setup: Beam radius of 1.76283 cm and wavefront curvature of -1763.79 m. So both results are in good agreement.

Since OSCAR 3.10, the thick lens can be made with an instance of the class Mirror. This example has been updated accordingly.

5.2 Calculating diffraction loss and circulating power

In this example, we will calculate the circulating power in a Fabry Perot cavity. At the same time, we can also derive the diffraction loss due to the finite size of the mirrors.

The Fabry Perot cavity is 1 km long and the input and end mirrors have a radius of curvature of 2500 m. Both mirrors have a transmission of 2% and no loss. The diameter of the mirror is set to 10 cm. The incident beam has a beam radius of 2 cm and a radius wavefront curvature of 2500 m.

The 7 lines of the script to define the cavity and do the calculations can be found in the listing 5.2. This script with additional commented lines can be found in the OSCAR3.0 folder under the name Example_Pcirc.m.

The script takes less than 10s to run on a modern laptop. After the calculation, the script *Display_Results(C1)* will return in the Matlab console:

Listing 5.2: Example of OSCAR script to calculate the circulating power

```
G1 = Grid(128,0.3);
E_input = E_Field(G1,'w',0.02,'R',-2500);

IM = Interface(G1,'RoC',2500,'CA',0.1,'T',0.02);
EM = Interface(G1,'RoC',2500,'CA',0.1,'T',0.02);

C1 = Cavity1(IM,EM,1000,E_input);
C1 = Cavity_Resonance_Phase(C1);
C1 = Calculate_Fields_AC(C1);

Display_Results(C1);
```

```
Found the phase for resonance in cavity: C1
  Power in the input beam:        1   [W]
  Circulating power:      49.2824   [W]
  Transmitted power:      0.98563   [W]
  Reflected power:    0.0125915   [W]
  Round trip losses:      36.3807   [ppm]
```

5.3 Scan of a misaligned cavity

Using the same cavity parameters as in the previous example, we will now tilt the end mirror in the vertical direction by 1 microradian. To see the excitation of higher optical modes, we will display the circulating power over one free spectral range. The script for that simulation is presented in the listing 5.3.

Listing 5.3: Example of OSCAR script to scan a cavity

```
G1 = Grid(128,0.3);
E_input = E_Field(G1,'w',0.02,'R',-2500);

IM = Interface(G1,'RoC',2500,'CA',0.1,'T',0.02);
EM = Interface(G1,'RoC',2500,'CA',0.1,'T',0.02);
EM = Add_Tilt(EM,1E-6,'y');

C1 = Cavity1(IM,EM,1000,E_input);
C1 = Cavity_Scan(C1);
Display_Scan(C1);
```

The script takes around 2 minutes to run. The output is the Matlab figure

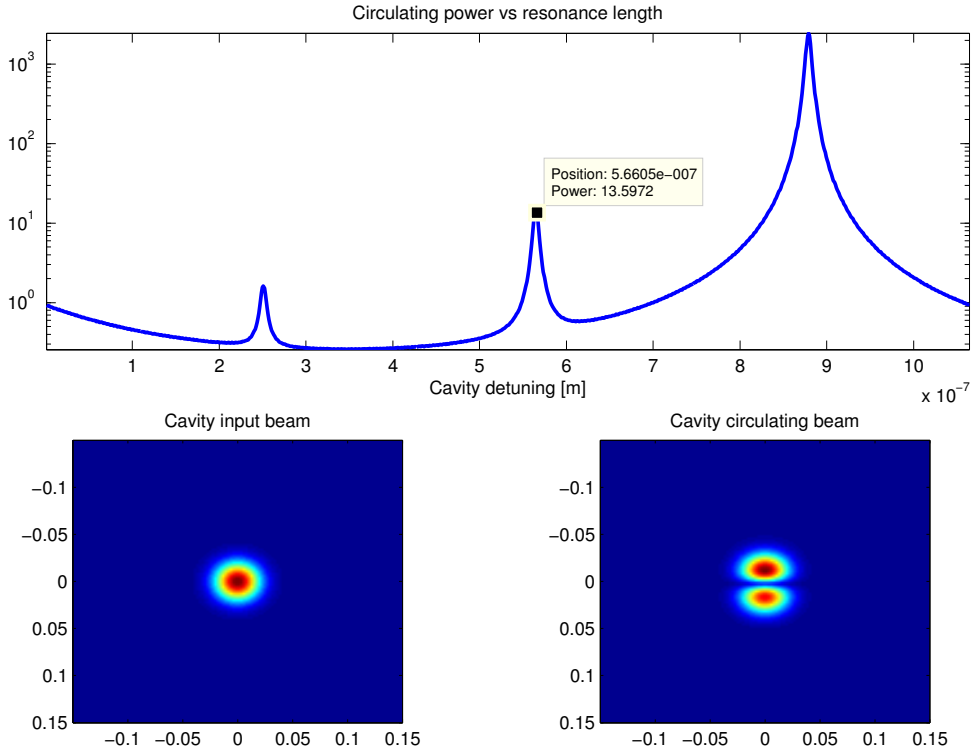


Figure 5.1: Output window of the function *Display_scan(C1)* with the cavity scan over one FSR on top.

shown in the figure 5.1. The top plot is a cavity scan over one free spectral range. We can notice 3 peaks : from left to right the first peak is a LG_{10} due to imperfect mode-matching, the second is the mode HG_{10} and the third and highest one is the fundamental mode. It is possible to move the cursor to see the shape of the circulating field for a particular detuning.

5.4 Mirror maps and higher order modes

In this example, we will demonstrate how to use higher order optical modes in conjunction with mirror maps. The mirror maps are stored in text files under the form of a matrix which represents the mirror height fluctuations in meters. In the maps given, the mirror curvature is not included (but it could be done). The RMS of the mirror maps is around 2 nm, typical of very good

large mirrors.

The laser beam is now defined outside the cavity, so the cavity reflected field can also be calculated. The script for this example is called Example_HOM_with_maps.m and is shown in the listing 5.4 without any comments.

Listing 5.4: Example of OSCAR script to calculate the reflected field from a cavity with maps

```
G1 = Grid(512,0.4);
E_input = E_Field(G1,'w',0.043,'R',−1034,'mode','LG_3_3');

IM = Interface(G1,'RoC',1500,'CA',0.4,'T',0.02);
EM = Interface(G1,'RoC',1700,'CA',0.4,'T',2E−6);

param_PSD = [0.02 −1.4];
Virtual_map_IM = Do_Virtual_Map(G1,param_PSD);
Virtual_map_EM = Do_Virtual_Map(G1,param_PSD);

IM = Add_Map(IM,Virtual_map_IM,'reso',G1.Step,'remove_tilt_focus',0.150,'RMS',1E−9,'verbose',false);
EM = Add_Map(EM,Virtual_map_EM,'reso',G1.Step,'remove_tilt_focus',0.150,'RMS',1E−9,'verbose',false);

C1 = Cavity1(IM,EM,3000,E_input);
C1.Propagation_mat.Use_DI = true;

C1 = Cavity_Resonance_Phase(C1);
C1 = Calculate_Fields_AC(C1);
Display_Results(C1);
```

The line `Virtual_map_IM = Do\_Virtual\_Map(G1,param_PSD);` is used to create a custom 2D height map to be later applied to the surface `IM`. The map does not have to have the same size as the grid, since internally an interpolation is done with `Add_Map()`. In that case the map is automatically normalized to have a RMS of 1 nm. Regarding the sign convention, positive values in the mirror maps reduce the length of the cavity (so equivalent to a bump on the surface).

After the definition of the cavity and the calculation of the resonance length, the command `C1 = Calculate_Fields_AC(C1)` is used to calculate the circulating, reflected and transmitted fields from the cavity. The fields are then stored in the instance `C1` where they could be accessed later. To display the result graphically the command `Display_Results(C1)` is then used. An example of the output screen is shown in figure 5.2

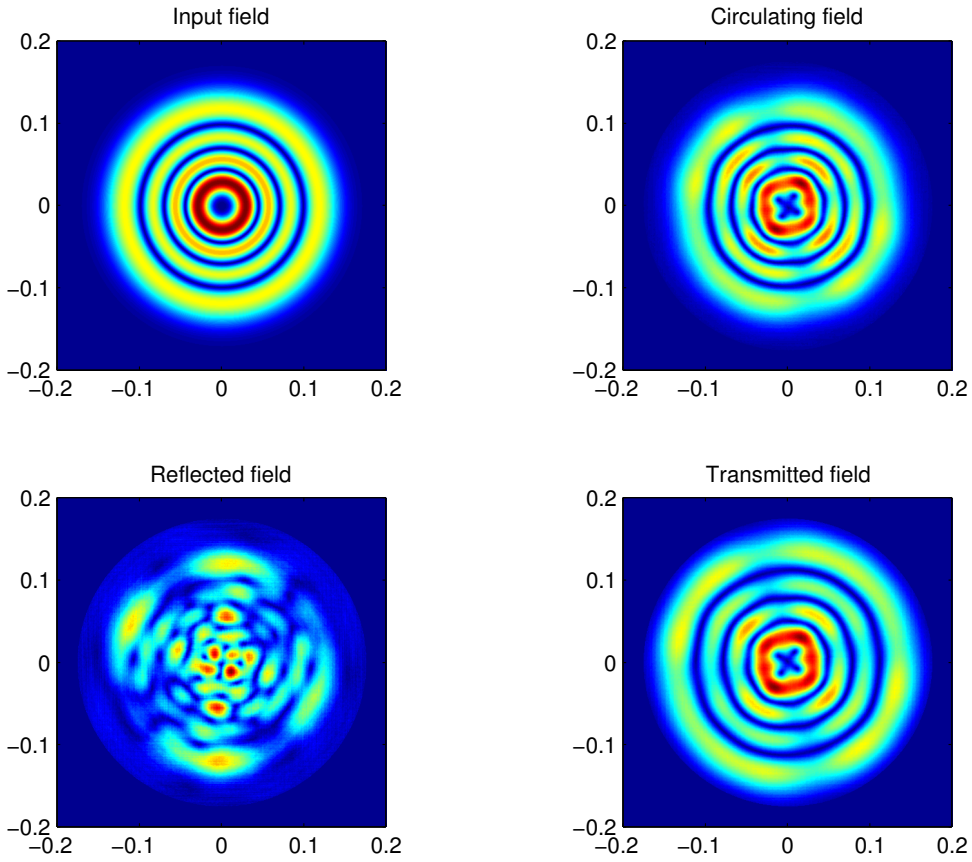


Figure 5.2: Output window of the function *Display_results(C1)* when a LG33 mode is expected to circulate in a cavity with realistic mirrors.

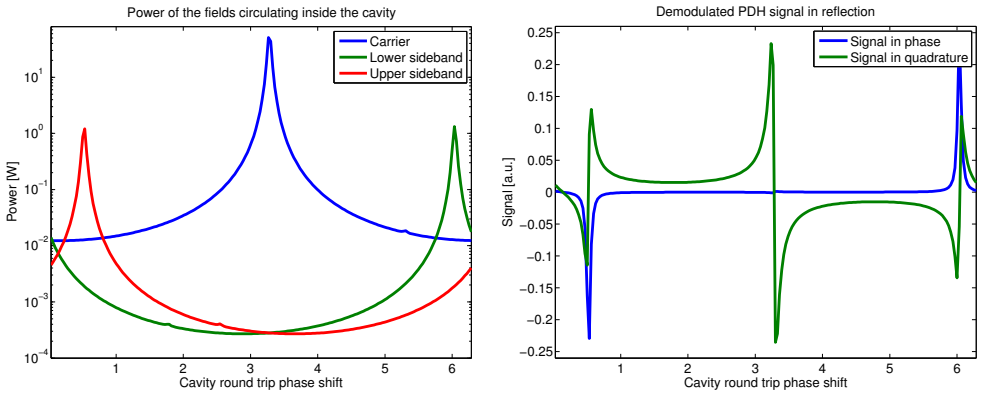


Figure 5.3: Circulating power inside the cavity (left) and error signals in reflection (right).

5.5 Plotting a Pound-Drever-Hall error signal

In this example, we will learn how to create sidebands and derive a Pound-Drever-Hall locking signal in reflection. After the input laser `E1` is defined, a virtual phase modulator creates a pair of sidebands with the command: `E_input = Add_Sidebands(E_input,'Mod_freq',6.7234E6,'Mod_depth',0.3)` (sideband frequency: 6.7234 MHz and modulation depth: 0.3 . Then the fields inside the cavity are computed for a given resonance phase.

To draw the PDH error signals, the reflected field from the cavity is demodulated to get the in phase and in quadrature signals. The simulation is relatively long since for the cavity scan 200 points are taken. That takes around 12 minutes on my laptop. A plot of the circulating power and the reflected PDH error signal is shown in figure 5.3.

5.6 4-mirror linear cavity

This example, introduced with OSCAR 3.05, shows how to simulate a 4 mirrors cavity. The parameters to define the cavity are taken from the GEO600 output mode-cleaner. The script file is called Example_4_mirrors.m and is rather self-explanatory for the regular user.

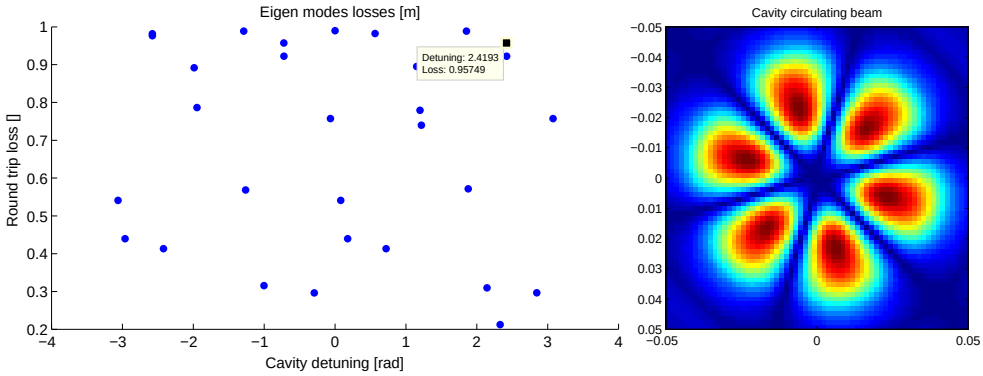


Figure 5.4: Result of the command Display_cavity_modes().

5.7 Calculating the eigenmodes of a Fabry-Perot cavity

Since OSCAR V3.10, it is possible to calculate the eigenmodes of a Fabry-Perot or any linear cavity such as a mode-cleaner. This particularly useful to check the degeneracy of a cavity or the clipping loss of higher order modes. The example file is called Example_cavity_eigenmodes.m.

A typical display of the calculation of the cavity eigenmodes is shown in figure 5.4. In this example file is also shown in the end how to display the Airy peak of the different higher order modes of the cavity.

Bibliography

[1] V. K. Madisetti and D. Williams, *The Digital Signal Processing Handbook*. CRC Press, 1998.

[2] http://en.wikipedia.org/wiki/Fourier_transform. A compact reminder for the main properties of the Fourier transform.

[3] M. Seul, L. O'Gorman, and M. J. Sammon, *Practical Algorithms for Image Analysis: Description, Examples, and Code*. Cambridge University Press, 2000. Some extensive illustrations of the results of the 2D Fourier transform on some simple patterns.

[4] H. Kogelnik and T. Li, "Laser beams and resonators," *Appl. Opt.*, vol. 5, pp. 1550–1567, 1966. A classic article at the foundation of many others, absolutely worth reading.

[5] W. T. Silfvast, *Laser Fundamentals*. Cambridge University Press, 2004. The section 11.2 of this book is a good reminder how to calculate the circulating field in an iterative way.

[6] B. Bochner, *Modelling the Performance of Interferometric Gravitational-Wave Detectors with Realistically Imperfect Optics*. PhD thesis, MIT, 1991.

[7] A. Tridgell, D. E. McClelland, and C. M. Savage, "Numerically modelling a dual recycling interferometric gravitational wave detector," in *Gravitational Astronomy: Instrument Design and Astrophysical Prospects*, World Scientific Publishing, (Singapore), 1991.

[8] J. Agresti, *Researches on Non-standard Optics for Advanced Gravitational Waves Interferometers.* PhD thesis, University of Pisa, 2007.

[9] A. G. Fox and T. Li, "Computation of optical resonator modes by the method of resonance excitation," *IEEE j. quantum electron*, vol. 4, pp. 460–465, 1968.

[10] http://www.sr.bham.ac.uk/dokuwiki/doku.php?id=geosim:oscar. GEO600 Simulation Group.

[11] Webpage of the package Finesse.

[12] P. Saha, "Fast estimation of transverse fields in high-finesse optical cavities," *J. Opt. Soc.Am. A*, vol. 14, no. 9, pp. 2195–2202, 1997.

[13] W. Koechner, *Solid-State Laser Engineering*. Springer Series in Optical Sciences, Springer, 2006.

[14] A. E. Siegman, *Lasers*. University Science Books, 1986. THE laser Bible, does it still need an introduction ?

[15] P. Barriga, B. Bhawal, L. Ju, and D. G. Blair, "Numerical calculations of diffraction losses in advanced interferometric gravitational wave detectors," *J. Opt. Soc.Am. A*, vol. 24, pp. 1731–1741, 2007.

[16] P. Hello and J. Y. Vinet, "Analytical models of thermal aberrations in massive mirrors heated by high power laser beams," *J. Phys. (France)*, vol. 51, pp. 1267–1282, 1990.

[17] M. Bondarescu1 and K. S. Thorne, "New family of light beams and mirror shapes for future LIGO interferometers," *Phys. Rev. D*, vol. 74, p. 082003, 2006.

[18] R. W. P. Drever, J. L. Hall, F. V. Kowalski, J. Hough, G. M. Ford, A. J. Munley, and H. Ward, "Laser phase and frequency stabilization using an optical resonator," *Applied Physics B: Lasers and Optics*, vol. 31, pp. 97–105, 1983.

[19] E. D. Black, "An introduction to Pound–Drever–Hall laser frequency stabilization," *American Journal of Physics*, vol. 69, pp. 79–87, 2001.

[20] J. Vinet, "The Virgo Physics Book, Vol. II." https://wwwcascina.virgo.infn.it/vpb/, 2006.

[21] H. Yamamoto, "Acceleration of the FFT simulation of stable cavity." LIGO-T0900640-v2, 2010.

[22] F. Shen and A. Wang, "Fast-Fourier-transform based numerical integration method for the Rayleigh-Sommerfeld diffraction formula," *Appl. Opt.*, vol. 45, pp. 1102–1110, 2006.

[23] H. Yamamoto, "1D PSD of mirror maps." LIGO-T1100353-v1, 2011.

[24] R. Day, "A new FFT code: FOG Fast Fourier Transform Optical Simulation of Gravitational Wave Interferometers." LIGO G1200629-v1, 2012.

[25] N. Melchert, J. Degallaix, L. Hinz, and E. Reithmeier, "GPU implementation of FSR simulations: performance improvements and limitations," in *Optics and Photonics for Information Processing XVI*, vol. 12225, p. 122250E, International Society for Optics and Photonics, SPIE, 2022.

Part III

Appendix

Appendix A

Analytical formulation of the Gaussian beam propagation using Fourier transform

In this appendix, we will see how it is possible to rediscover all the formulae related to the laser beam propagation just by using the Fourier technique explained in chapter 1. Basically, we will do by hand the three steps which are numerically done in Matlab to propagate a Gaussian beam: first, do a 2D Fourier transform of the laser beam, then add the phase shift in the frequency domain and finally, take the inverse Fourier transform.

To facilitate the mathematical calculation, we will extensively use the following formula:

$$\int_{\infty}^{+\infty} \exp\left(-ax^2 + ibx\right) dx = \sqrt{\frac{\pi}{a}} \exp\left(\frac{-b^2}{4a}\right) \tag{A.1}$$

Let us consider a Gaussian beam $E(x, y, z)$ of amplitude A. For simplification we suppose the waist w_0 of the beam to be located at $z = 0$. The beam can be written:

$$E(x, y, 0) = A \exp\left(-\frac{x^2 + y^2}{w_0^2}\right) \tag{A.2}$$

We will now propagate the Gaussian beam along the z axis over a distance d. To not complicate the calculation, we will just consider one transverse dimension at first, for example the dimension along x since all the following equations can be decoupled in the x/y dimensions.

The first step is then to calculate the Fourier transform $\widetilde{E}(\nu_x, 0)$ of the field $E(x, 0)$:

$$\widetilde{E}(\nu_x, 0) = \int_{\infty}^{+\infty} A \exp\left(-\frac{x^2}{w_0^2}\right) \exp(j2\pi\nu_x x) dx \tag{A.3}$$

Using the formula A.1, the above equation becomes:

$$\widetilde{E}(\nu_x, 0) = A\sqrt{\pi w_0^2} \exp\left(-(\pi\nu_x w_0)^2\right) \tag{A.4}$$

To simulate the propagation the electric field over a distance d, the proper phase shift is added to Fourrier transform:

$$\widetilde{E}(\nu_x, d) = A\sqrt{\pi w_0^2} \exp\left(-(\pi\nu_x w_0)^2\right) \exp(-jkd + j\lambda\pi\nu_x^2 d) \tag{A.5}$$

Finally, we take the inverse Fourier transform of the equation A.5 to go back to the familiar x space coordinate system:

$$\begin{aligned} E(x, d) &= \int_{\infty}^{+\infty} A\sqrt{\pi w_0^2} \exp\left(-(\pi\nu_x w_0)^2\right) \exp(-jkd + j\lambda\pi\nu_x^2 d) \exp(-j2\pi\nu_x x) d\nu_x \\ &= A\sqrt{\pi w_0^2} \exp(-jkd) \int_{\infty}^{+\infty} \exp\left(-(\pi^2 w_0^2 - j\lambda\pi d)\nu_x^2 - j2\pi x\nu_x\right) d\nu_x \\ &= A\sqrt{\pi w_0^2} \sqrt{\frac{\pi}{\pi^2 w_0^2 - j\pi\lambda d}} \exp(-jkd) \exp\left(\frac{-4\pi^2 x^2}{4(\pi^2 w_0^2 - j\lambda\pi d)}\right) \\ &= A\sqrt{\frac{1}{1 - j\frac{\lambda d}{\pi w_0^2}}} \exp(-jkd) \exp\left(-\frac{x^2}{w_0^2\left(1 - j\frac{\lambda d}{\pi w_0^2}\right)}\right) \end{aligned} \tag{A.6}$$

We can now define the usual Raleigh range z_r as:

$$z_r = \frac{\pi w_0^2}{\lambda} \tag{A.7}$$

By inserting the Raleigh range z_r into A.6 and by adding the similar result on the other y transverse dimension, we obtain:

$$E(x, y, d) = A\frac{1}{1 - j\frac{d}{z_r}} \exp(-jkd) \exp\left(-\frac{x^2 + y^2}{w_0^2\left(\frac{1}{1 - j\frac{d}{z_r}}\right)}\right) \tag{A.8}$$

Let us analyze now the two main constituents of equation A.8, the exponential and the complex factor in front of it. The complex number in the exponential can be separated in its real and imaginary parts:

$$\begin{aligned}
\exp\left(-\frac{x^2+y^2}{w_0^2\left(\frac{1}{1-j\frac{d}{z_r}}\right)}\right) &= \exp\left(-\frac{x^2+y^2}{w_0^2\left(1+\frac{d^2}{z_r^2}\right)} - j\frac{(x^2+y^2)\frac{d}{z_r}}{w_0^2\left(1+\frac{d^2}{z_r^2}\right)}\right) \\
&= \exp\left(-\frac{x^2+y^2}{w_0^2\left(1+\frac{d^2}{z_r^2}\right)} - j\frac{(x^2+y^2)}{w_0^2\left(\frac{z_r}{d}+\frac{d}{z_r}\right)}\right) \\
&= \exp\left(-\frac{x^2+y^2}{w_0^2\left(1+\frac{d^2}{z_r^2}\right)} - j\frac{(x^2+y^2)}{\frac{2z_r}{k}\left(\frac{z_r}{d}+\frac{d}{z_r}\right)}\right) \\
&= \exp\left(-\frac{x^2+y^2}{w_0^2\left(1+\frac{d^2}{z_r^2}\right)} - jk\frac{(x^2+y^2)}{2d\left(1+\frac{z_r^2}{d^2}\right)}\right)
\end{aligned} \tag{A.9}$$

We can now consider the factor in front equation A.8:

$$A\frac{1}{1-j\frac{d}{z_r}} = A\left(\frac{1}{1+\frac{d^2}{z_r^2}}\right)^{\frac{1}{2}}\exp\left(j\arctan\left(\frac{d}{z_r}\right)\right) \tag{A.10}$$

Combining equations A.9 and A.10, we can deduce the familiar equations governing the propagation of Gaussian beams:

$$E(x,y,d) = A\frac{w_0}{w(d)}\exp\left(-\frac{x^2+y^2}{w(d)^2}\right)\exp\left(-jk\left(d+\frac{x^2+y^2}{2R(d)}\right)+j\arctan\left(\frac{d}{z_r}\right)\right) \tag{A.11}$$

With:

$$\begin{aligned}
w(d) &= w_0\left(1+\frac{d^2}{z_r^2}\right) \\
R(d) &= d\left(1+\frac{z_r^2}{d^2}\right)
\end{aligned} \tag{A.12}$$

I found remarkable that by using the simple FFT steps to propagate the beam we can rediscover all the usual equations: the normalisation factor, the evolution of the beam radius and the wavefront radius of curvature as well as the Gouy phase shift. The courageous readers can also do the calculations presented here for the higher order modes in the Hermite-Gauss base.

Appendix B

Finesse script

The Finesse script [11] used to simulate a Fabry Perot cavity equivalent to the one described in the table 2.1 is presented below. It is often extremely useful to create some simple Finesse model to check the exactitude of OSCAR results whenever possible. The script is also included in the OSCAR distribution under the name P_circ.kat in the folder Calculate_Pcirc.

```
l i1 1 0 n0                              # laser P=1W
gauss G1 i1 n0 0.017221 -517.112239      # Parameters for a beam radius 2cm
                                         # and a wavefront RofC of -2000m

s s0 1n 1 n0 n1
m1 SITM1 1 0 0 n1 n2
s sITM2 1n 1.45 n2 nITM
m1 ITM2 0.005 50E-6 0 nITM ncav1         # IM transmission and loss
attr ITM2 Rc -2000

s s3 1000 ncav1 ncav2

m1 ETM 50E-6 50E-6 0 ncav2 n4            # EM transmission and loss
attr ETM Rc 2000

cav cav_mode ITM2 ncav1 ETM ncav2

maxtem 8
trace 10
phase 0
startnode n0
```

```
pd0 p_circ ncav1
pd0 p_trans n4
pd0 p_ref n1

xaxis ETM phi lin -233 -234 3600
yaxis abs

gnuterm NO
retrace off
```

Appendix C

Details of OSCAR 3.0 classes

The properties (i.e. parameters included in an object of a class) of the different classes are described in following sections. Properties are not protected, so they can be accessed and changed using: object_name.properties (at your own risk).

The name of the properties are usually self-explanatory and derived directly from OSCAR classic.

C.1 Class Grid

C.2 Class E_Field

C.3 Class Prop_operator

C.4 Class Interface

C.5 Class Mirror

C.6 Class Cavity1

C.7 Class CavityN

Table C.1: Details of the properties of the class *Grid*

Properties	Type	Comments
Num_point	scalar, integer	Should be a power of 2
Length	scalar	Length of one side of the grid (m)
Step	scalar	Step = Length / Num_point
Half_num_point	scalar, integer	Half_num_point = Num_point / 2
Vector	vector, integer	from 1 to Num_point
Axis	vector	The scale for the grid
Axis_FFT	vector	The spatial frequency scale
D2_X	2D matrix	Horizontal scale in 2D
D2_Y	2D matrix	Vertical scale in 2D
D2_square	2D matrix	Square of the distance to the center
D2_r	2D matrix	Distance from the center of the grid
D2_FFT_X	2D matrix	Horizontal FFT scale in 2D
D2_FFT_Y	2D matrix	Vertical FFT scale in 2D

Table C.2: Details of the properties of the class *E_Field*

Properties	Type	Comments
Grid	Grid	Grid where the beam is defined
Field	2D complex matrix	The carrier electric field
Field_SBl	2D complex matrix	The lower sidebands
Field_SBu	2D complex matrix	The higher sidebands
Refractive_index	scalar	Refractive index of the medium
Wavelength	scalar	The light wavelength
Frequency_Offset	scalar	Frequency of the sidebands
Mode_name	string	Family and mode number
k_prop	scalar	Propagation constant

Table C.3: Details of the properties of the class *Prop_operator*

Properties	Type	Comments
n	scalar	Refractive index of the media
mat	2D complex	Propagation matrix
dist	scalar	Distance of propagation
Use_DI	logical	Enable digital integration
mat_DI	2D complex	Propagation matrix for digital integration

Table C.4: Details of the properties of the class *Interface*

Properties	Type	Comments
Grid	Grid	Grid where the beam is defined
surface	2D matrix	The height of the surface
mask	2D matrix of 0 and 1	The aperture of the optic
T	scalar	Transmission in power
L	scalar	Loss in power
n1	scalar	The first refractive index
n2	scalar	The second refractive index
t	scalar	Amplitude transmission $\sqrt{T}$
r	scalar	Amplitude reflectivity $\sqrt{1-(T+L)}$

Table C.5: Details of the properties of the class *Mirror*

Properties	Type	Comments
I_HR	Interface	First surface
I_AR	Interface	Second surface
length_substrate	scalar	Length of the substrate
RT_inside	integer	Number of round-trips
n_substrate	scalar	Refractive index of the substrate
r	scalar	Reflectivity

Table C.6: Details of the properties of the class *Cavity1*

Properties	Type	Comments
I_input	Interface	Input mirror surface
I_end	Interface	End mirror surface
Length	scalar	Length of the cavity
Laser_in	E_Field	Input laser beam
Laser_start_on_input	logical	Define where the input beam is given
Resonance_phase	complex scalar	Phase adjustment to bring the cavity on resonance
Cavity_scan_all_field	vector of E_Field	Store all the E_field after each round-trip
Cavity_scan_param	vector	3 values to define how to scan the cavity
Cavity_phase_param	scalar	Number of round-trips used to find the resonance
Cavity_scan_R	vector	Cavity circulating power scan over one FSR
Cavity_scan_RZ	vector	Zoom of the scan around the cavity resonance
Cavity_EM_mat	2D complex matrix	Cavity round-trip kernel
Propagation_mat	Prop_operator	Pre-computed propagation matrix
Field_circ	E_Field	Cavity circulating field
Field_ref	E_Field	Cavity reflected field
Field_trans	E_Field	Cavity transmitted field

Table C.7: Details of the properties of the class *CavityN*

Properties	Type	Comments
I_array	Vector Interface	Mirror surfaces
d_array	Vector Prop_operator	Distance between mirrors
Nb_mirror	Integer	Number of mirrors
Laser_in	E_Field	Input laser beam
Laser_start_on_input	logical	Define where the input beam is given
Resonance_phase	complex scalar	Resonance cavity tuning
Cavity_scan_all_field	vector of E_Field	Store all the E_field after each round-trip
Cavity_scan_param	vector	3 values to define how to scan the cavity
Cavity_phase_param	scalar	Nb of round-trips used to find the resonance
Cavity_scan_R	vector	Store the FSR scan
Cavity_scan_RZ	vector	Zoom of the scan around the cavity resonance
Cavity_E_mat	2D complex matrix	Cavity round-trip kernel
Propagation_mat_array;	Vector of array	Pre-computed propagation matrices
Field_circ	E_Field	Cavity circulating field
Field_ref	E_Field	Cavity reflected field
Field_trans	E_Field	Cavity transmitted field

Appendix D

They have found OSCAR useful...

... and they mention it. A list of publications where OSCAR is quoted:

D.1 Thesis

1. Degallaix, J. (2006). Compensation of strong thermal lensing in advanced interferometric gravitational waves detectors, University of Western Australia). *Editor note: Of course, when it all started.*

2. Granata, M. (2011). Optical development for second-and third-generation gravitational-wave detectors: stable recycling cavities for advanced virgo and higher-order Laguerre-Gauss modes, Université Paris Diderot and Università degli Studi di Roma Tor Vergata.

3. Bonnand, R. (2012). The Advanced Virgo gravitational wave detector: Study of the optical design and development of the mirror, Université Claude Bernard Lyon I

4. Fang, Q. (2015). High Optical Power Experiments and Parametric Instability in 80 m Fabry-P erot Cavities, University of Western Australia.

5. Straniero, N. (2015). Étude, développement et caractérisation des miroirs des interféromètres laser de 2ème génération dédiés à la détection des ondes gravitationnelles, Université Claude Bernard-Lyon I

6. Ott, K. (2016). Towards a squeezing-enhanced atomic clock on a chip, Université Pierre et Marie Curie.

7. Favier, P. (2017). Etude et conception d'une cavité Fabry-Perot de haute finesse pour la source compacte de rayons X ThomX, Paris Saclay

8. Blair, C. (2017). Parametric Instability in Gravitational Wave Detectors (Doctoral dissertation, University of Western Australia

9. Metzdorff, R. (2019). Refroidissement de résonateurs mécaniques macroscopiques proche de leur état quantique fondamental, Laboratoire Kastler Brossel

10. Hardwick, T. (2019). High Power and Optomechanics in Advanced LIGO Detectors, Louisiana State University

D.2 Articles

1. Barriga, P., Bhawal, B., Ju, L., & Blair, D. G. (2007). Numerical calculations of diffraction losses in advanced interferometric gravitational wave detectors. JOSA A, 24(6), 1731-1741. *Editor note: at that time the code has no name.*

2. Crouzil, T., & Perrin, M. (2013). Dynamics of a chain of optically coupled micro droplets, J. Europ. Opt. Soc. Rap. Public. 8, 13079

3. Gatto, A., Tacca, M., Kéfélian, F., Buy, C., & Barsuglia, M. (2014). Fabry-Pérot-Michelson interferometer using higher-order Laguerre-Gauss modes. Physical Review D, 90(12), 122011.

4. Zhao, C., Ju, L., Fang, Q., Blair, C., Qin, J., Blair, D., ... & Yamamoto, H. (2015). Parametric instability in long optical cavities and suppression by dynamic transverse mode frequency modulation. Physical Review D, 91(9), 092001.

5. Straniero, N., Degallaix, J., Flaminio, R., Pinard, L., & Cagnoli, G. (2015). Realistic loss estimation due to the mirror surfaces in a 10 meters-long high finesse Fabry-Perot filter-cavity. Optics express, 23(16), 21455-21476.

6. Allocca, A., Gatto, A., Tacca, M., Day, R. A., Barsuglia, M., Pillant, G., ... & Vajente, G. (2015). Higher-order Laguerre-Gauss interferometry for gravitational-wave detectors with in situ mirror defects compensation. Physical Review D, 92(10), 102002.

7. Ott, K., Garcia, S., Kohlhaas, R., Schüppert, K., Rosenbusch, P., Long, R., & Reichel, J. (2016). Millimeter-long fiber Fabry-Perot cavities. Optics express, 24(9), 9839-9853.

8. Capocasa, E., Barsuglia, M., Degallaix, J., Pinard, L., Straniero, N., Schnabel, R., ... & Flaminio, R. (2016). Estimation of losses in a 300 m filter cavity and quantum noise reduction in the KAGRA gravitational-wave detector. Physical Review D, 93(8), 082004.

9. Blair, C., Gras, S., Abbott, R., Aston, S., Betzwieser, J., Blair, D., ... & Grote, H. (2017). First demonstration of electrostatic damping of parametric instability at Advanced LIGO. Physical review letters, 118(15), 151102.

10. Wittmuess, P., Piehler, S., Dietrich, T., Ahmed, M. A., Graf, T., & Sawodny, O. (2016). Numerical modeling of multimode laser resonators. JOSA B, 33(11), 2278-2287.

11. Ma, Y., Liu, J., Ma, Y. Q., Zhao, C., Ju, L., Blair, D. G., & Zhu, Z. H. (2017). Thermal modulation for suppression of parametric instability in advanced gravitational wave detectors. Classical and Quantum Gravity, 34(13), 135001.

12. Capocasa, E., Guo, Y., Eisenmann, M., Zhao, Y., Tomura, A., Arai, K., ... & Somiya, K. (2018). Measurement of optical losses in a high-finesse 300 m filter cavity for broadband quantum noise reduction in gravitational-wave detectors. Physical Review D, 98(2), 022010.

13. Jia, Y., Huang, R., Lan, Y., Ren, Y., Jiang, H., & Lee, D. (2019). Reversible Aggregation and Dispersion of Particles at a Liquid–Liquid Interface Using Space Charge Injection. Advanced Materials Interfaces, 6(5), 1801920.

14. Hardwick, T., Hamedan, V. J., Blair, C., Green, A. C., & Vander-Hyde, D. (2020). Demonstration of dynamic thermal compensation for parametric instability suppression in Advanced LIGO. Classical and Quantum Gravity.

15. Wu, B., Blair, C., Ju, L., Zhao, C,. (2020) Contoured thermal deformation of mirror surface for detuning parametric instability in an optical cavity. Classical and Quantum Gravity, 37(12):125003.

16. Wang, H., Amoudry, L., Cassou, K., Chiche, R., Degallaix, J., Dupraz, K., Huang, W., Martens, A., Michel, C., Monard, H., Nutarelli, D. (2020) Prior-damage dynamics in a high-finesse optical enhancement cavity. Applied Optics, 59(35):10995-1002.

Acknowledgements

I would like to thank all the users of the code and especially all the people who have helped in the development, in particular : Massimo Galimberti (impressive work for the Zernike implementation and the ZYGO .dat file interpreter), Francois Bondu (for generating maps from a 1D PSD), Luc Di Gallo, Francois Labaye and Richard Day for all the interesting discussions.

The GPU coding was initiated, pushed and later optimised by Nils Melchert.

Aline Cahuzac corrected the mistakes in this manual, the remaining ones were added later by me.