OCMS

Optical Cavity Mode Solver for 2D Cavities

Quick Guide

Document Version 201101
November 1, 2020

Relevant Products:
  OCMS-2020-Basic
  OCMS-2020-Extension-Tools

Telecognix Corporation

Telecognix Corporation

Japan Head Office:
Yoshida Shimooji-cho 58-13
Sakyo-ku, Kyoto 606-8314
Tel: +81-75-762-4633
Fax: +81-75-762-4631

Kyoto Business Office:
612 Stork Bld. Sanjo Karasuma,
Kamanza-cho 22, Nakagyo-ku, Kyoto 604-8241
Tel: +81-75-213-3599
Fax: +81-75-213-3599

OCMS Technical Support Contact:
ocms@telecognix.com

Table of contents

*** The tools introduced in Sections 9 and 10 (i.e., batch_plot.py and autofinder.py) are only included in the Extension-Tools package.**

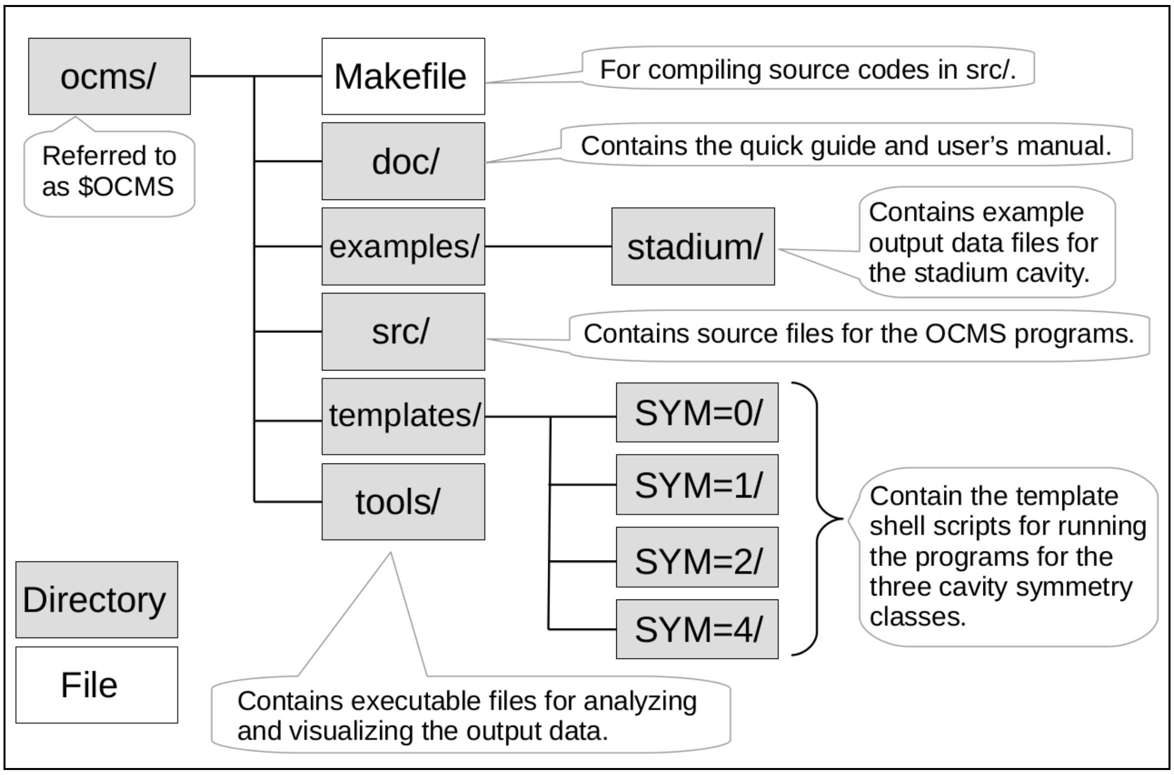
1. Introduction

The OCMS program package offers the following three functions:

- Detection of the resonances for a two-dimensional cavity.
- Computation and plotting of the wave function of a resonant mode.
- Computation and plotting of the Husimi distribution of a resonant mode.

This quick guide demonstrates the procedures for using the above functions by introducing a specific simulation example. By following the steps shown in this quick guide, the reader should be able to acquire the basic usage and structure of the programs contained in the OCMS package.

Figure 1.1 shows the directory structure of the OCMS package with brief descriptions of each directory.



ocms/
Referred to as $OCMS
Makefile
For compiling source codes in src/.
doc/
Contains the quick guide and user's manual.
examples/
stadium/
Contains example output data files for the stadium cavity.
src/
Contains source files for the OCMS programs.
templates/
SYM=0/
SYM=1/
SYM=2/
SYM=4/
Contain the template shell scripts for running the programs for the three cavity symmetry classes.
tools/
Contains executable files for analyzing and visualizing the output data.
Directory
File

Fig. 1.1: Directory structure of the OCMS package.

Go back to table of contents

2. Prerequisites and preparations

2.1. Supported OS

- Ubuntu LTS version 18.04 or later

2.2. Software installation

Before using the OCMS programs, the following software tools and libraries need to be installed:

- gfortran
- LAPACK and BLAS
- SLATEC
- Python 3 ··· for plotting wave functions and Husimi distributions
  Caution: The Python examples in this quick guide will trigger the default Python 2.7 interpreter unless Python 3 is properly configured as explained in the User's Manual.
- gnuplot ··· for plotting far-field and near-field patterns

See the appendices of User's Manual for how to install these tools.

2.3. Path setting

Set $ocms as the the path to the OCMS's home directory:

```
$ export ocms=the/path/to/ocms/
```

For example, if you have the ocms directory under /home/foo/, you must set ocms=/home/foo/ocms.

Add $ocms/tools to PATH:

```
$ export PATH=$PATH:$ocms/tools
```

It is useful to set the above paths in the shell script such as .bashrc and .zshrc in your home directory so that these settings persist.

2.4. Preparation of a work space directory

For running executable files and outputting data files, the user is required to make their own work space directory wherever they want. In what follows, we use $workspace as the path to this directory.

Although it is not required, for following this quick guide, it is convenient to set workspace as an environmental variable as follows:

```
$ export workspace=the/path/to/workspace
```

Go back to table of contents

3. Simulation conditions

In this quick guide, we demonstrate how to find resonances and how to plot their wave functions and Husimi distributions, taking the following simulation conditions as an example:

- Cavity shape: Stadium with R=L=1 where R is radius of curvature of curved sections and L is half-length of straight section (R and L can be specified independently in the cavity model, but all the examples provided here use the default setting R=L for simplicity).

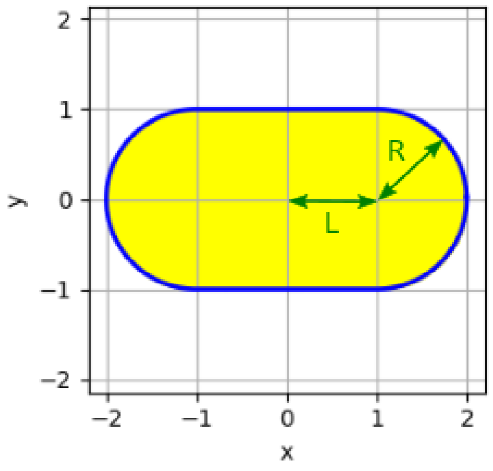


Fig. 3.1: Stadium cavity.

- Refractive indices: nin=3.3 (inside the cavity), nout=1.0 (outside the cavity)
- Wave number range: around k=5
- Polarization: TM (transverse magnetic)
- Parity: even-even (a=b=1)   [see Appendix 1 for the definition of the parity]

In the following steps, the reader will create output data files by themselves in the directory $workspace. The output files and shell scripts related to the simulations in this quick guide can also be found in $ocms/examples/stadium/.

Go back to table of contents

4. Setting the parameters and building executable files

4.1. Setting the cavity shape parameters in def_cavity…F

The codes for defining the cavity shape are found in $ocms/src/def_cavities:

```
$ ls -1 $ocms/src/def_cavities
def_cavity_SYM0_Sinai.F
def_cavity_SYM0_asym_cut_disk.F
def_cavity_SYM0_asym_limacon.F
def_cavity_SYM0_rounded_triangle.F
def_cavity_SYM1_D-shape.F
def_cavity_SYM1_annular.F
def_cavity_SYM1_cardioid.F
def_cavity_SYM2_D2_deformed_circle.F
def_cavity_SYM2_Sinai.F
def_cavity_SYM2_ellipse.F
def_cavity_SYM2_flattened_quadrupole.F
def_cavity_SYM2_stadium.F
def_cavity_SYM4_Sinai.F
```

The file names are labeled by the symmetry class parameter SYM (=0, 1, 2 or 4). We have chosen the stadium as the cavity shape. So we use the code "def_cavity_SYM2_stadium.F". The stadium belongs to the symmetry class labeled by SYM=2, because it is symmetric with respect to both x and y axes (see Appendix 1 for the definition of SYM).

The shape parameters for the stadium can be set in the subroutine "set_parameters" defined in the beginning of def_cavity_SYM2_stadium.F.

```
*
subroutine set_parameters()
*
implicit NONE
*
double precision R,L
common /const/ R,L
*
R=1d0*(SIZE_PARAM)
L=1d0*(SIZE_PARAM)
*
return
end
```

By default, the shape parameters R and L are set as R=L=1d0*(SIZE_PARAM), where SIZE_PARAM is the cavity size parameter defined in "build-params.mk" explained below (the default SIZE_PARAM value is 1d0). For the simulations demonstrated in this quick guide, we fix the stadium shape with

R=L=1d0 (i.e., SIZE_PARAM=1d0).

If the stadium cavity of your interest has R=L=10 [μm], you can set SIZE_PARAM=10d0, and consider that all the spatial variables have the dimension μm, and the wave number has the dimension 1/μm.

Alternatively, using the fact that the Helmholtz equation solely depends on the dimensionless parameter nkR (n, k and R are the refractive index, wave number, and characteristic length, respectively), you can fix the values of R and L to be 1, and instead scale all the quantities with an appropriate factor. Let us suppose that the stadium cavity of your interest has the radius R'=30 [μm] and the half-length of the flat part L'=30 [μm]. Then, the scale factor becomes

$$\alpha = R'/R = L'/L = 30\ [\mu m].$$

When interpreting the simulation data obtained for the stadium with R=L=1, you just need to multiply this scale factor to the spatial variables. For example, x=0.5 for the stadium cavity with R=L=1 means $x'=\alpha\times x=15$ [μm] for the stadium cavity with R'=L'=30 [μm]. Meanwhile, the wave number is scaled by $k'=k/\alpha$ (wave number has the dimension of inverse length). For example, k=1.0 for the stadium cavity with R=L=1 corresponds to $k'=1/(30\ [\mu m]) \approx 0.33\ [1/\mu m]$.

Go back to table of contents

4.2. Setting fundamental parameters in build-params.mk

The fundamental parameters that are used for building executable files are defined in the file “build-params.mk”. The other parameters that can be flexibly changed after the builds are defined in the shell scripts for running the executable files (i.e., det.sh, wfunc.sh, and husimi.sh). The parameters of the OCMS programs are summarized in Appendix 2.

Let us first look at build-params.mk:

```
#
# build-params.mk
#

#======================
#   Cavity symmetry
#======================
SYM = 2 # The cavity is symmetric with respect to both x- and y-axis

#===================
#   Cavity shape
#===================
#CAVITY = def_cavity_SYM2_D2_deformed_circle.F
#CAVITY = def_cavity_SYM2_Sinai.F
```

```
#CAVITY = def_cavity_SYM2_ellipse.F
#CAVITY = def_cavity_SYM2_flattened_quadrupole.F
CAVITY = def_cavity_SYM2_stadium.F

#============================
#   Cavity size parameter
#============================
SIZE_PARAM = 1d0

#==============================
#   Polarization (TM or TE)
#==============================
PLZ = TM
#PLZ = TE

#======================================
#   The number of boundary elements
#======================================
NBE = 50

#=======================================================
#   Grid points for plotting the Husimi distribution
#   (parameters for main.husimi.F)
#=======================================================
IXMAX = 200
IYMAX = 200
```

build-params.mk is called by Makefile when we build executable files. Templates of this file can be found in $ocms/templates/SYM{0,1,2,4}. We first copy this template to the work space:

```
cp $ocms/templates/SYM2/build-params.mk $workspace
```

Here we chose SYM2, because SYM=2 for the stadium cavity. In build-params.mk, we set the parameter values as follows:

- SYM=2
- CAVITY = def_cavity_SYM2_stadium.F
- SIZE_PARAM=1d0
- PLZ=TM

IXMAX and IYMAX are parameters for the Husimi distribution plot (described in Section 8). IXMAX and IYMAX are the numbers of grid points for the x and y coordinates, and set as IXMAX = IYMAX = 200 by default. When you need better resolution for the Husimi distribution plot, increase these values.

As for the parameter NBE, we must set an appropriate value. NBE defines the number of boundary elements for the fundamental boundary segment (for SYM=4, the fundamental boundary segment (FBS) is the boundary curve in the first quadrant with x>y, for SYM=2, the FBS is the boundary curve in the first quadrant (as shown in Fig. 5.1 (top right) for the stadium case), for SYM=1, the FBS is the boundary curve in the upper half plane, and for SYM=0, the FBS is the whole boundary curve).

In the next section, we describe how we find an appropriate NBE value.

Go back to table of contents

4.3. Finding an appropriate value for NBE

NBE must be determined so that the boundary elements sufficiently “resolve” the half-wavelength (inside the cavity) of the resonant mode in the wavelength regime of interest. Let’s say we are interested in the wavelength range around λ. Then, this condition reads

$$\text{ratio} := \lambda/(2 \times \text{nin} \times \text{ds\_max}) = \pi/(\text{nin} \times k \times \text{ds\_max}) >> 1,$$

where nin is the refractive index inside the cavity, ds_max the maximum length of the boundary elements, and k the wave number. ds_max gets smaller as we increase NBE (ds_max is roughly inversely proportional to NBE). For a given k, we need to find an NBE value that satisfies the above condition. In principle, a larger NBE value results in better accuracy. However, it also requires longer computation time. In practice, the ratio around 3 results in minimally acceptable accuracy and reasonable computation time (REMARK: If you need high accuracy in your research, you should set the ratio much larger than 3).

The program “estimateNBE” helps the user to find an appropriate NBE value.

Compilation of estimateNBE:

```
cd $ocms
make estimateNBE params=$workspace/build-params.mk
```

This make command only refers to variable “CAVITY” in build-params.mk, which is set as “CAVITY = def_cavity_SYM2_stadium.F” here. This command creates estimateNBE in the directory $ocms.

Execution of estimateNBE:

Running estimateNBE without command-line arguments outputs the following message:

```
$ ./estimateNBE
Error: insufficient or too many command-line arguments.

Usage: estimateNBE [nin] [k] [ratio]

nin : Refractive index inside the cavity.
k : Wave number (in the vacuum).
ratio := (lambda_in/ds_max)/2,
where lambda_in is the wave length inside the cavity,
and ds_max is the maximum length of the boundary element.
```

For given nin, k and ratio, estimateNBE returns the minimum NBE-value that satisfies

$$\lambda/(2 \times \text{nin} \times \text{ds\_max}) \geq \text{ratio}.$$

The following is the result of estimateNBE, when nin=3.3, k=5.0 and ratio=3.0:

```
$ ./estimateNBE 3.3 5.0 3.0
NBE=41 --> ratio=3.030303
```

This result indicates that NBE must be 41 or larger so that ratio exceeds 3.0.

There is also a case when the user wants to know the value of ratio for given nin, k and NBE. For this case, the program “estimateRatio” can be used.

Compilation of estimateRatio:

```
cd $ocms
make estimateRatio params=$workspace/build-params.mk
```

This make command only refers to the variable “CAVITY” in build-params.mk, which is set as “CAVITY = def_cavity_SYM2_stadium.F” here. This command creates estimateRatio in the directory $ocms.

Execution of estimateRatio:

Running estimateRatio without command-line arguments outputs the following message:

```
$ ./estimateRatio
Error: insufficient or too many command-line arguments.

Usage: estimateRatio [nin] [k] [NBE]

nin : Refractive index inside the cavity.
k : Wave number (in the vacuum).
NBE : The number of boundary elements.
```

The following is the result of estimateRatio when nin=3.3, k=5.0 and NBE=50:

```
$ ./estimateRatio 3.3 5.0 50
--> 0.5*lambda/ds_max = 3.64
```

We will fix NBE to be 50 in the simulations described below.

Go back to table of contents

4.4. Building executable files

With the determination of the NBE value in the previous section, we are ready for building all the executable files. We assume that you have prepared build-params.mk (including the setting of NBE) put in $workspace, and that you are now in the directory $ocms:

```
$ cd $ocms
```

You can create executable files by the following command:

```
$ make cavity det det_dirichlet wfunc husimi params=$workspace/build-params.mk
```

Then, you get the five executable files, cavity_data, det.TM.NBE=50, det_dirichlet.TM.NBE=50, wfunc.TM.NBE=50, and husimi.TM.NBE=50. Alternatively, you can use “all” option:

```
$ make all params=$workspace/build-params.mk
```

This creates estimateNBE and estimateRatio in addition to the above five executable files. All the executable files will be run under the work space. So we move them to $workspace/bin.

```
cd $workspace
mkdir bin
mv   $ocms/cavity_data   $ocms/det.TM.NBE=50   $ocms/det_dirichlet.TM.NBE=50
$ocms/wfunc.TM.NBE=50 $ocms/husimi.TM.NBE=50 ./bin
```

Go back to table of contents

5. Output the cavity boundary data

We assume that you are in the directory $workspace:

```
$ cd $workspace
```

Run the executable file named “cavity_data” in the directory bin/:

```
$ ./bin/cavity_data
```

Then, you get the two output files, “data.cavity_boundary” and “data.cavity_domain”, which contain the following information:

- data.cavity_boundary : data for the cavity boundary points, normal vectors, the length of boundary elements, and curvatures.

- data.cavity_domain : data for the cavity domain.

These data files can be visualized by using “boundary_plot.py” in the directory $ocms/tools as shown below. Also, “data.cavity_boundary” will be later used when we plot a wave function.

Visualize the cavity boundary data:

```
$ boundary_plot.py --boundary data.cavity_boundary --domain
data.cavity_domain
```

This command displays various information regarding the cavity boundary as shown in Fig. 5.1. The functions of the top bar buttons are described in Fig. 5.2. For closing the window, push the “q” button.

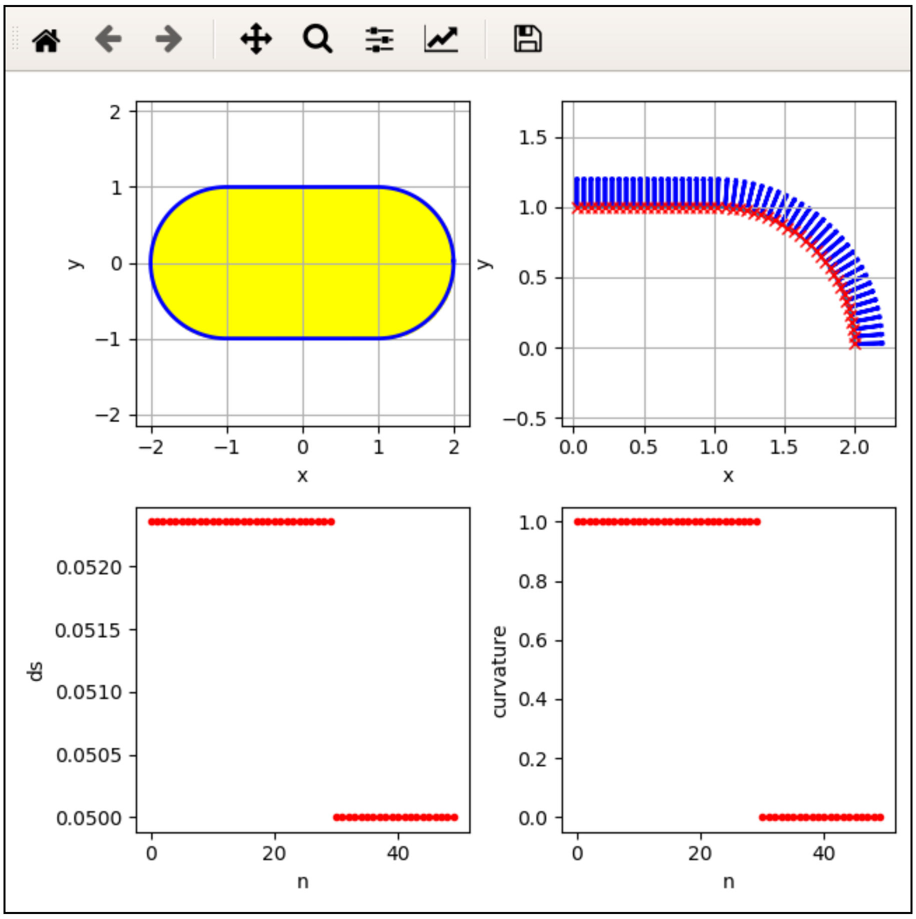


Fig. 5.1: (Top left) The cavity domain defined by data.cavity_domain (yellow region) and the cavity boundary curve defined by data.cavity_boundary (blue curve); (Top right) The cavity boundary points (red crosses) with normal vectors (blue lines); (Bottom left) The dependence of the length of a boundary element on the cavity boundary element index n; (Bottom right) The dependence of the curvature on the cavity boundary element index n.

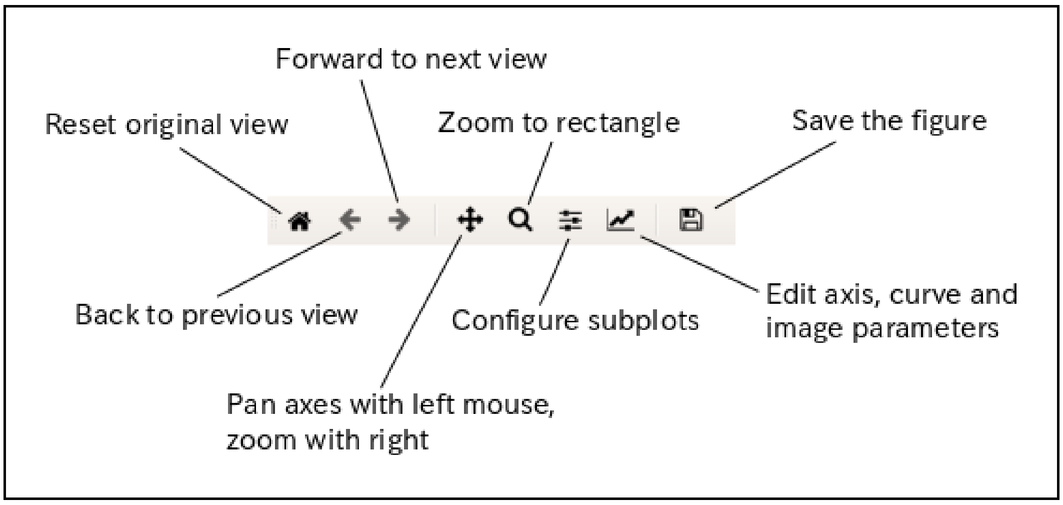


Fig. 5.2: The functions of the top bar buttons.

Go back to table of contents

6. Search and detect the resonances

6.1. Global search

For resonance search we use the executable file “det.TM.NBE=50”, which can be run by the shell script “det.sh”. By setting the parameter values in this shell script, you can specify the resonance search domain in the complex wave number space as shown in Fig. 6.1.

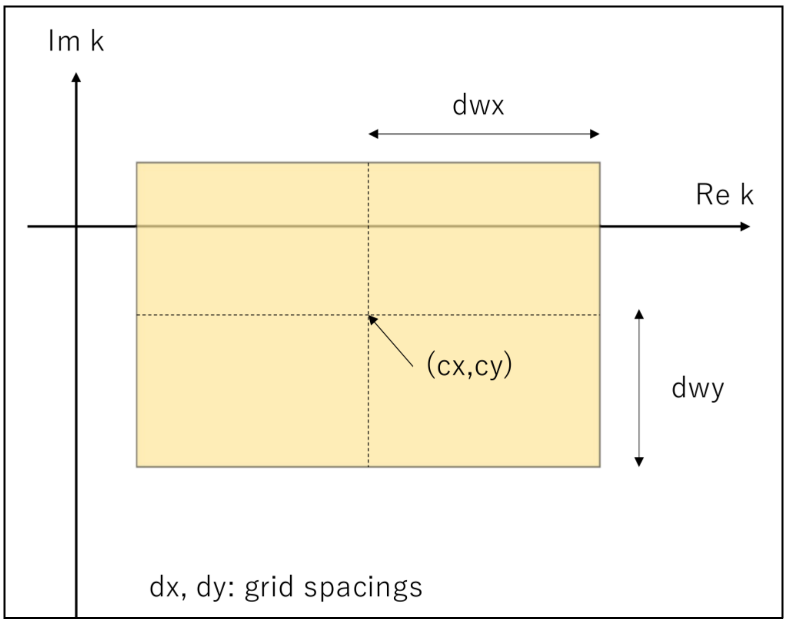


Fig. 6.1: Resonance search domain in the complex wave number space.

Here we choose (cx,cy)=(5.0,-0.03), dwx=0.1, and dwy=0.03. Also, the grid spacings for the resonance search are set to be dx=0.01 and dy=0.01.

There is a template file for det.sh under the directory $ocms/templates/SYM{0,1,2,4}. The template file is different for different symmetry classes. For the stadium cavity, you must use the one in the SYM2 directory.

For resonance search, it is recommended to make a new directory “resonance_search” under $workspace, and run det.TM.NBE=50 from this directory.

```
$ cd $workspace
$ mkdir resonance_search
$ cd resonance_search
$ cp $ocms/templates/SYM2/det.sh .
```

Go back to table of contents

6.1.1. Parameter setting in det.sh:

Open det.sh by an editor, and input the parameter values as follows:

```
#!/bin/bash -f
#
#   det.sh for the cavity with symmetry class SYM=2
#
#   Shell script for running the the executable file det.[TM/TE].NBE=...
#   for calculating the determinant value distribution for a given
#   resonance search domain in the complex wave number space.
#
#   See ocms/doc/users_guide.pdf for the details.
#
#================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___S_T_A_R_T_S___H_E_R_E
#================================================================

# Executable file name (e.g., ./det.TM.NBE=50)
execfile=../bin/det.TM.NBE=50

# Refractive indices inside and outside the cavity
nin=3.3d0
nout=1d0

# Parity indices (a,b = -1(odd) or +1(even))
a=1
b=1

# Resonance search domain:
#  (cx,cy) : center of the search domain
#  dwx,dwy : the width of the search domain
#  dx,dy   : the grid spacings
cx=5.0d0
cy=-0.03d0

dwx=0.1d0
dwy=0.03d0

 dx=0.01d0
 dy=0.01d0

#================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___E_N_D_S___H_E_R_E
#================================================================

# Output data file name (DO NOT TOUCH)
outputfile="dat.det.cx="$cx".cy="$cy."dx="$dx."dy="$dy
```

```
# Run the execfile (DO NOT TOUCH)
echo $nin $nout $a $b $cx $cy $dwx $dwy $dx $dy | $execfile > $outputfile
```

Go back to table of contents

6.1.2. Run det.sh:

```
$ ./det.sh
```

It takes about 20 sec to finish on a PC with a Core i7-4800MQ 2.7GHz processor. You get the output file “dat.det.cx=5.0d0.cy=-0.03d0.dx=0.01d0.dy=0.01d0”.

6.1.3. Analyze the output data

```
$ minfinder.py dat.det.cx=5.0d0.cy=-0.03d0.dx=0.01d0.dy=0.01d0 --annotate
```

This command opens a window showing the (scaled) determinant value distribution in the complex wave number space as shown in Fig. 6.2. In addition, the command shows the list of local minimums to the standard output:

```
$ minfinder.py dat.det.cx=5.0d0.cy=-0.03d0.dx=0.01d0.dy=0.01d0 --annotate
5.01 -0.02 0.29810383
5.09 -0.02 0.30028649
5.06 -0.01 0.29310035
```

The first column of the output data represents the real part of the wave number, the second column the imaginary part of the wave number, and the third column the scaled determinant value (the scaling is automatically carried out in the program for avoiding underflow). In the above result, three resonances are found at (5.01, -0.02), (5.09, -0.02), and (5.06, -0.01).

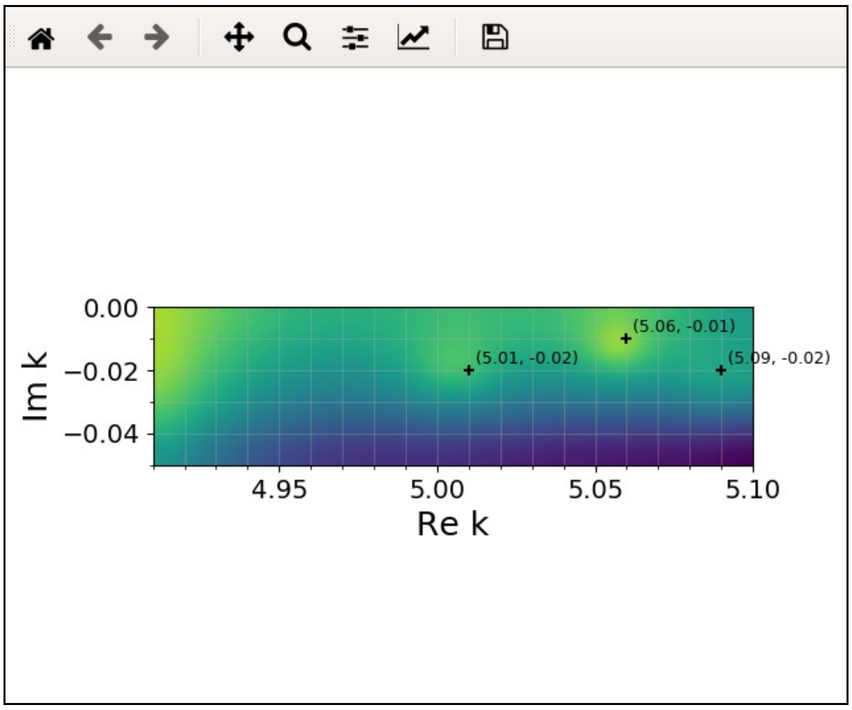


Fig. 6.2: Distribution of the (scaled) determinant values in the complex wave number space. The plus (+) represents the local minimum of the distribution, corresponding to a resonance.

Go back to table of contents

6.2. Local search

In the above resonance search, the resolution of the resonant wave number was limited by the search grid spacings dx=dy=0.01. For gaining better resolution, you need to set finer grid spacings. Here let us limit our attention to one of the resonances found at (5.06, -0.01), and determine its position more precisely.

6.2.1. Parameter resetting of det.sh:

In det.sh, we change the parameter values for the resonance search domain. This time, the search domain is centered at (cx,cy) = (5.06, -0.01) with dwx=dwy=0.01 and dx=dy=0.001. Note that dx and dy values of the previous search (i.e., dx=dy=0.01) are now input to dwx and dwy.

```
#!/bin/bash -f
#
#   det.sh for the cavity with symmetry class SYM=2
#
#   Shell script for running the the executable file det.[TM/TE].NBE=...
#   for calculating the determinant value distribution for a given
#   resonance search domain in the complex wave number space.
#
#   See ocms/doc/users_guide.pdf for the details.
#
#================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___S_T_A_R_T_S___H_E_R_E
#================================================================

# Executable file name (e.g., ./det.TM.NBE=50)
execfile=../bin/det.TM.NBE=50

# Refractive indices inside and outside the cavity
nin=3.3d0
nout=1d0

# Parity indices (a,b = -1(odd) or +1(even))
a=1
b=1

# Resonance search domain:
#  (cx,cy) : center of the search domain
#  dwx,dwy : the width of the search domain
#  dx,dy   : the grid spacings
cx=5.06d0
cy=-0.01d0

dwx=0.01d0
dwy=0.01d0

 dx=0.001d0
 dy=0.001d0

#================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___E_N_D_S___H_E_R_E
#================================================================

# Output data file name (DO NOT TOUCH)
outputfile="dat.det.cx="$cx".cy="$cy."dx="$dx."dy="$dy

# Run the execfile (DO NOT TOUCH)
echo $nin $nout $a $b $cx $cy $dwx $dwy $dx $dy | $execfile > $outputfile
```

6.2.2. Run det.sh

```
$ ./det.sh
```

It takes about 1 min to finish on a PC with a Core i7-4800MQ 2.7GHz processor. You get the output file “dat.det.cx=5.06d0.cy=-0.01d0.dx=0.001d0.dy=0.001d0”.

Go back to table of contents

6.2.3. Analyze the output data

```
$ minfinder.py dat.det.cx=5.06d0.cy=-0.01d0.dx=0.001d0.dy=0.001d0 --annotate
5.059 -0.009 0.28806639
```

This command opens a window showing the determinant value distribution as shown in Fig. 6.3. The resonance is found at (5.059, -0.009).

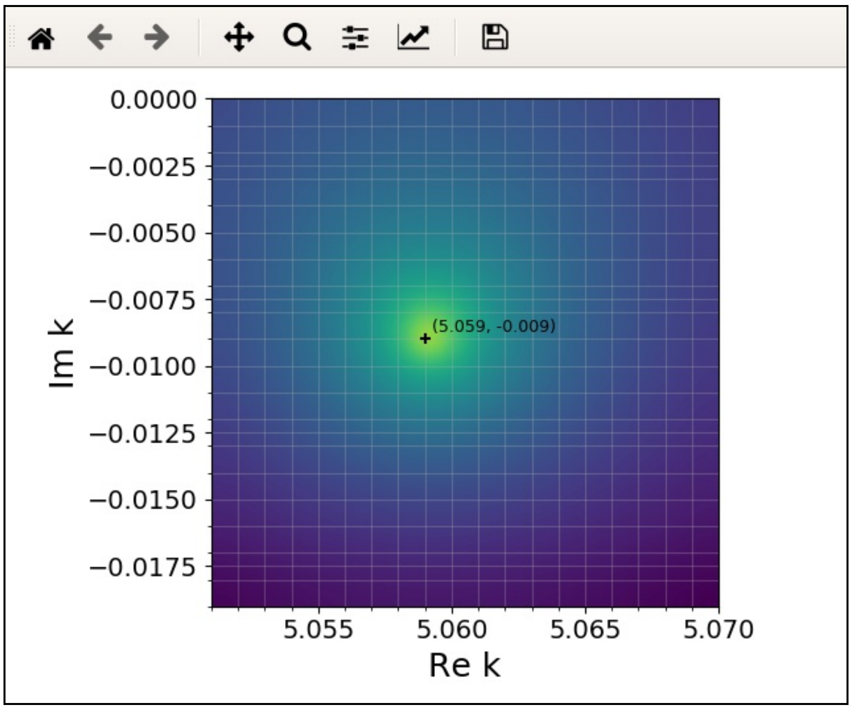


Fig. 6.3: Distribution of the determinant values around (5.06,-0.01).

6.2.4. Iteration of the refining process

The precision of the resonant wave number can be further increased by iterating the similar procedure. Here we omit demonstrating this iteration process, and just state the result. For the local search with dx=dy=1e-5, the resonance is found at k = (5.05914, -0.00876). Below, we use this k value for plotting a wave function and a Husimi distribution.

Go back to table of contents

6.3 How to distinguish spurious modes

The boundary integral equation for a dielectric cavity contains spurious solutions, which correspond to the solutions of the interior Dirichlet problem with refractive index 1, i.e.,

$$\left[\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+k^2\right]\psi(x,y)=0 \quad \text{with } \psi=0 \text{ for } (x,y)\in\partial D$$

where ∂D represents the cavity boundary. In this section, a method to distinguish spurious modes is described. The basic strategy is to additionally solve the interior Dirichlet problem to find which modes are spurious.

Figure 6.4 shows a distribution of the determinant values in the complex wave number space (the computation conditions are given in the figure caption). In Fig. 6.4, we find a low-loss resonance at k=5.18+i0.0. This mode will turn out to be a spurious mode, by additionally solving the interior Dirichlet problem with refractive index 1.

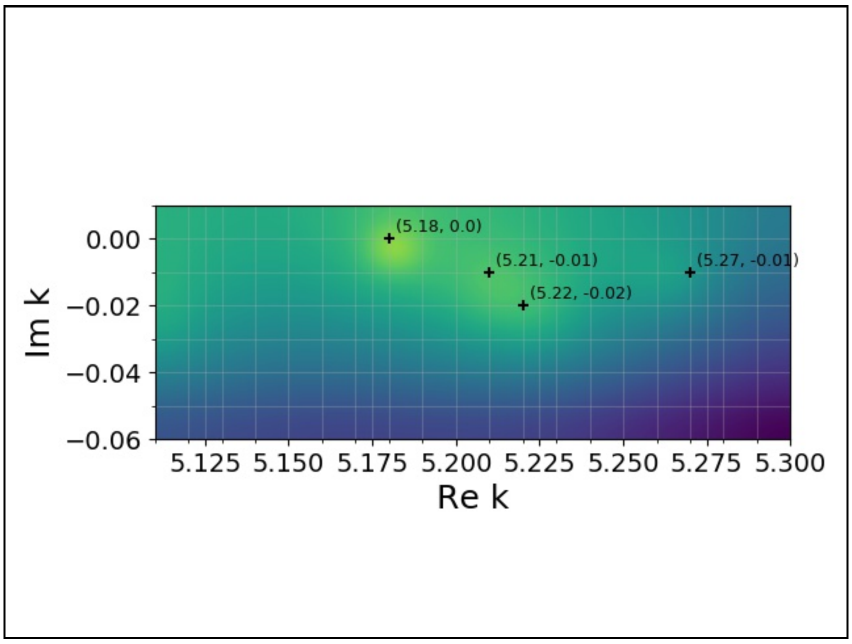


Fig. 6.4: Distribution of the determinant values for a stadium cavity. The computation conditions are as follows: SIZE_PARAM=1, PLZ=TM, nin=3.3, nout=1.0, a=-1, b=1, NBE=100, cx=5.2, cy=-0.03, dwx=0.1, dwy=0.04, dx=0.01, dy=0.01.

In the OCMS package, the determinant for the interior Dirichlet problem with refractive index 1 can be computed by the executable *det_dirichlet.{TM/TE}.NBE=**. This file can be built by the following command:

```
$ make det_dirichlet
```

For the *build-params.mk* file in Sect. 4.2, we obtain the executable *det_dirichlet.TM.NBE=50*. This can be run by the script det_dirichlet.sh given below. This is a slight modification of *det.sh* for *det.TM.NBE=50*. Namely, “det.TM.NBE=50” is replaced by “det_dirichlet.TM.NBE=50” for the variable *execfile*, and the variables *nin* and *nout* are not used for the interior Dirichlet problem.

```
#!/bin/bash -f
#
#   det.sh for the cavity with symmetry class SYM=2
#
#   Shell script for running the the executable file det.[TM/TE].NBE=...
#   for calculating the determinant value distribution for a given
#   resonance search domain in the complex wave number space.
#
#   See ocms/doc/users_guide.pdf for the details.
#
#=================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___S_T_A_R_T_S___H_E_R_E
#=================================================================

# Executable file name (e.g., ./det.TM.NBE=50)
execfile=../bin/det_dirichlet.TM.NBE=50

# Refractive indices inside and outside the cavity
#nin=3.3d0
#nout=1d0

# Parity indices (a,b = -1(odd) or +1(even))
a=-1
b=1

# Resonance search domain:
#  (cx,cy) : center of the search domain
#  dwx,dwy : the width of the search domain
#  dx,dy   : the grid spacings
cx=5.2d0
cy=-0.03d0

dwx=0.1d0
dwy=0.04d0

 dx=0.01d0
 dy=0.01d0

#=================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___E_N_D_S___H_E_R_E
#=================================================================

# Output data file name (DO NOT TOUCH)
outputfile="dat.det.cx="$cx".cy="$cy."dx="$dx."dy="$dy
```

```
# Run the execfile (DO NOT TOUCH)
echo $nin $nout $a $b $cx $cy $dwx $dwy $dx $dy | $execfile > $outputfile
```

This script outputs the data file: *dat.det.cx=5.2d0.cy=-0.03d0.dx=0.01d0.dy=0.01d0*, which contains determinant values for the interior Dirichlet problem with refractive index 1. The following command displays the determinant value distribution:

```
$ minfinder.py dat.det.cx=5.2d0.cy=-0.03d0.dx=0.01d0.dy=0.01d0 --annotate
```

The result is shown in Fig. 6.5.

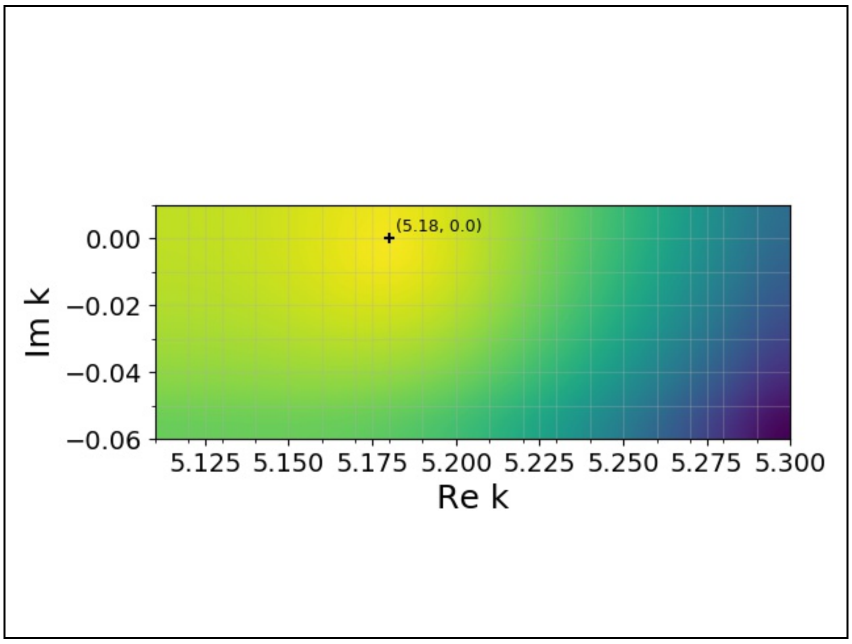


Fig. 6.5: Distribution of the determinant values of the interior Dirichlet problem with refractive index 1 for a stadium cavity.

This result indicates that k=5.18+i0.0 is an eigen wave number of the interior Dirichlet problem with refractive index 1. Therefore, we can conclude that the mode with k=5.18+i0.0 is spurious for the dielectric cavity problem.

TIPS: In the above script *det_dirichlet.sh*, the variables cy and dwy are set to the same values as in *det.sh*. However, considering that the eigen wave numbers of the interior Dirichlet problem are close to the real axis (theoretically, they should be exactly on the real axis), cy and dwy can be optimized, say, cy=-0.01, dwy=0.01. This optimization reduces computation time.

Go back to table of contents

7. Compute and plot the wave function

Here we demonstrate the computation and plotting of the wave function for the resonant mode with wave number k = 5.05914 - 0.00876 i. The execution file is wfunc.TM.NBE=50, and you need the shell script wfunc.sh to run it.

7.1. Create a directory for data output

```
$ cd $workspace
$ mkdir mode.ee.k=5.05914-0.00876i
```

Here “ee” means that the mode has the even-even parity (see Appendix 1 for the possible parity patterns). The directory name is arbitrary, but it has to uniquely specify the resonant mode like the above example name.

All the data files related to this mode, such as wave function, far-field, near-field, and Husimi distribution data files, will be output to this directory.

7.2. Copy wfunc.sh template

```
$ cd mode.ee.k=5.05914-0.00876i
$ cp $ocms/templates/SYM2/wfunc.sh .
```

Go back to table of contents

7.3. Parameter setting for wfunc.sh

Open wfunc.sh by an editor, and input the parameter values as follows:

```
#!/bin/bash -f
#
#   wfunc.sh for the cavity with symmetry class SYM=2
#
#   Shell script for running the executable file wfunc.[TM/TE].NBE=...
#   for calculating the wave function for a given wave number.
#
#   See ocms/doc/users_guide.pdf for the details.
#
#================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___S_T_A_R_T_S___H_E_R_E
#================================================================

# Executable file name (e.g., ../wfunc.TM.NBE=50)
execfile=../bin/wfunc.TM.NBE=50

# Refractive indices inside and outside the cavity
nin=3.3d0
nout=1d0

# Parity indices (a,b = -1(odd) or +1(even))
a=1
b=1

# Complex wave number of a resonant mode, k = kx + i ky
kx=5.05914d0
ky=-0.00876d0

# x and y ranges for outputting the data
xmin=-2.5d0
xmax=2.5d0
ymin=-1.5d0
ymax=1.5d0

# x and y coordinates grid points
ixmax=500
iymax=300

#================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___E_N_D_S___H_E_R_E
#================================================================

# Run the execfile (DO NOT TOUCH)
echo $nin $nout $a $b $kx $ky $xmin $xmax $ymin $ymax $ixmax $iymax
| $execfile
```

Go back to table of contents

7.4. Run wfunc.sh

```
$ ./wfunc.sh
```

It takes about 2 min to finish on a PC with a Core i7-4800MQ 2.7GHz processor. You get the following three data files regarding the wave function:

- dat.wfunc : wave function (can be plotted by plot2d.py)
- dat.farfield : far-field pattern (can be plotted by gnuplot)
- dat.nearfield : near-field pattern (can be plotted by gnuplot)

Go back to table of contents

7.5. Plot the wave function data

```
$ plot2d.py dat.wfunc --boundary ../data.cavity_boundary --linewidth 3
```

This command opens a window showing the intensity distribution |φ(x,y)|^2 as shown in Fig. 7.1 for the wave function data in dat.wfunc (this file stores the values of Re(φ) and Im(φ)).

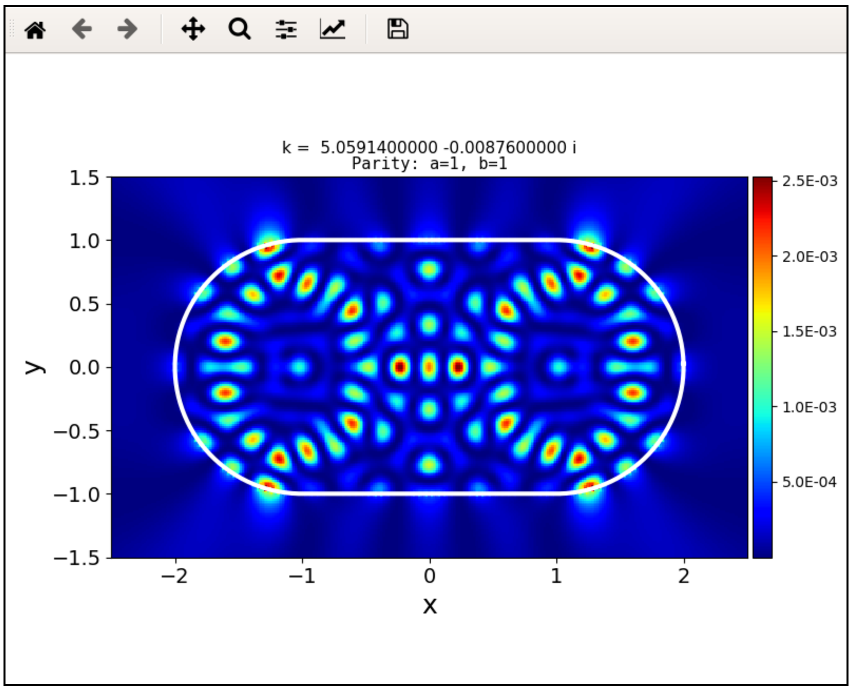


Fig. 7.1: The intensity distribution |φ(x,y)|^2 of the resonant mode with wave number k = 5.05914 - 0.00876 i.

The intensity (i.e., the vertical axis) is shown in the log scale with the option – –vscale log.

```
$ plot2d.py dat.wfunc --boundary ../data.cavity_boundary --linewidth 3
--vscale log
```

The result is shown in Fig. 7.2.

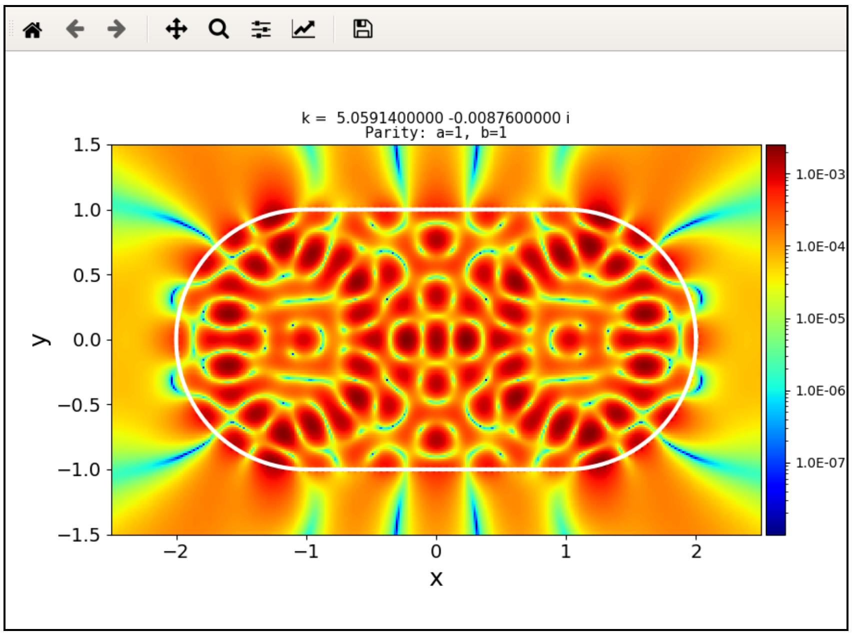


Fig. 7.2: Same as Fig. 7.1 but in log scale.

For adding units to the x and y variables, Matplotlib's interface can be used. By pushing the second top button from the right, a new window for figure options opens, where you can change the label from "x" to "x [μm]", for example.

The far-field and near-field pattern data are stored respectively in dat.farfield and dat.nearfield. They can be plotted by the gnuplot scripts, farfield.plt and nearfield.plt contained in $ocms/tools:

```
$ farfield.plt
$ nearfield.plt
```

farfield.plt and nearfield.plt respectively generate dat.farfield.png and dat.nearfield.png. These images are shown in Figs. 7.3 and 7.4.

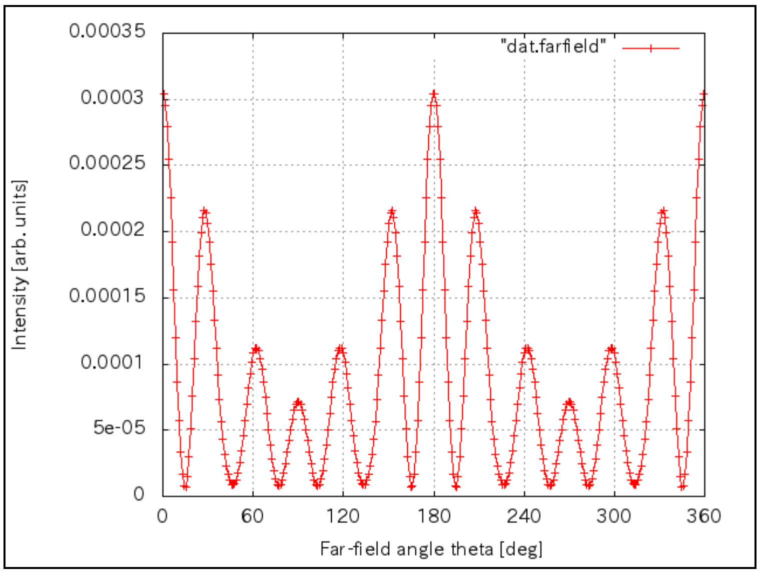


Fig. 7.3: Far-field emission pattern (i.e., the dependence of the absolute square of the scattering amplitude on the polar angle) of the resonant mode shown in Fig. 7.1.

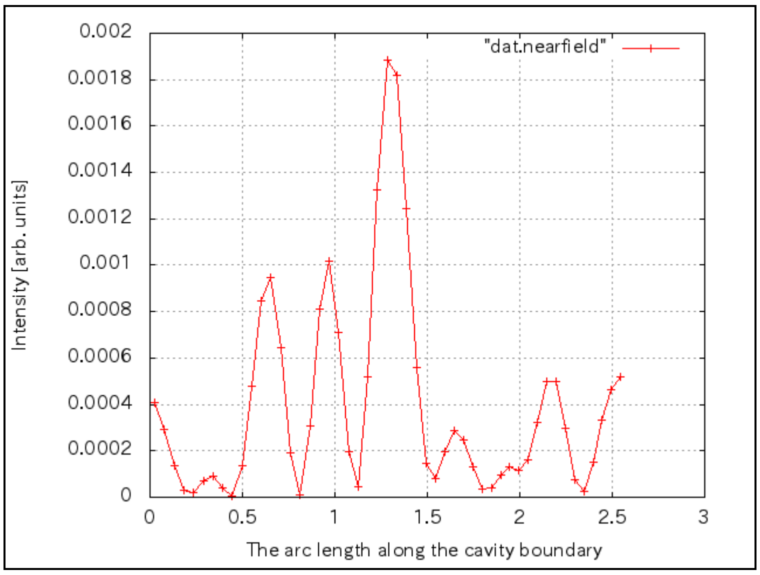


Fig. 7.4: Near-field pattern (i.e., the field intensity distribution along the cavity boundary) of the resonant mode shown in Fig. 7.1. The arc length is measured from the boundary point (x,y)=(2,0) to the boundary point (x,y)=(1,0) [See Fig. 5.1 (top left panel)]. The number of points coincides with NBE.

[Go back to table of contents](#)

8. Compute and plot the Husimi distribution

Here we demonstrate the computation and plotting of the Husimi distribution for the resonant mode with wave number k = 5.05914 - 0.00876 i. The execution file is husimi.TM.NBE=50, and you need the shell script husimi.sh to run it.

8.1. Copy husimi.sh template

```
cd $workspace/mode.ee.k=5.05914-0.00876i
cp $ocms/templates/SYM2/husimi.sh .
```

8.2. Parameter setting

Open husimi.sh by an editor, and input the parameter values as follows:

```
#!/bin/bash -f
#
#   husimi.sh for the cavity with symmetry class SYM=2
#
#   Shell script for running the executable file husimi.[TM/TE].NBE=...
#   for calculating the Husimi distribution for a given wave number.
#
#   See ocms/doc/users_guide.pdf for the details.
#
#=================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___S_T_A_R_T_S___H_E_R_E
#=================================================================

# Executable file name (e.g., ../husimi.TM.NBE=50)
execfile=../bin/husimi.TM.NBE=50

# Parity indices (a,b = -1 or +1)
nin=3.3d0
nout=1d0

# Parity indices (a,b = -1(odd) or +1(even))
a=1
b=1

# Complex wave number of a resonant mode, k = kx + i ky
kx=5.05914d0
ky=-0.00876d0
```

```
#=================================================================
#   P_A_R_A_M_E_T_E_R___S_E_T_T_I_N_G___E_N_D_S___H_E_R_E
#=================================================================

# Run the execfile (DO NOT TOUCH)
echo $nin $nout $a $b $kx $ky | $execfile
```

Go back to table of contents

8.3. Run husimi.sh

```
$ ./husimi.sh
```

It takes about 30 sec to finish on a PC with a Core i7-4800MQ 2.7GHz processor. This command outputs the following data file:

- dat.husimi : Husimi distribution (can be plotted by plot2d.py)

8.4. Plot the Husimi distribution

```
$ plot2d.py dat.husimi
```

This command opens a window showing the Husimi distribution as shown in Fig. 8.1.

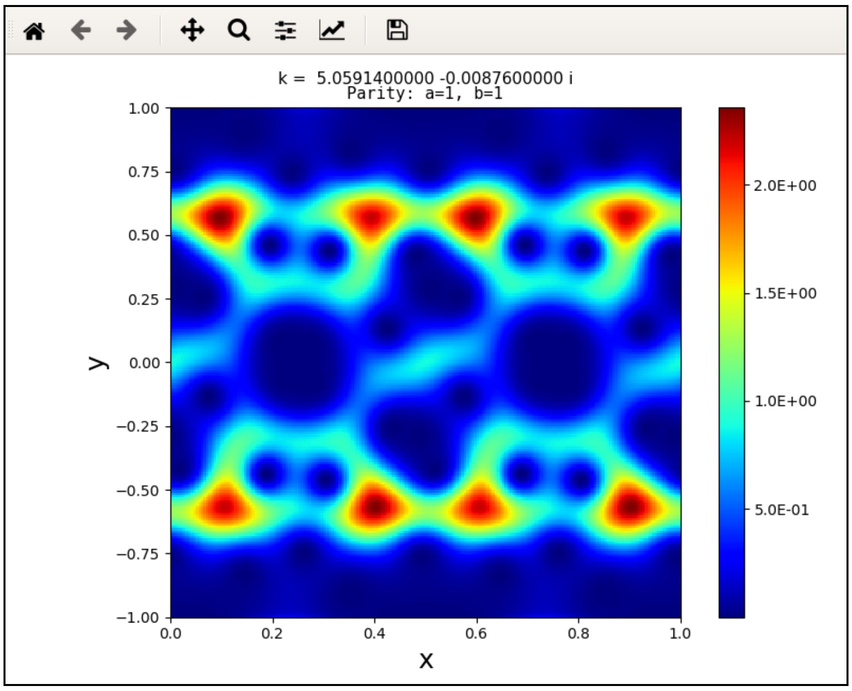


Fig. 8.1: The Husimi distribution of the resonant mode with wave number k = 5.05914 - 0.00876 i.

As mentioned in Section 4.2, the numbers of the x- and y-coordinate grid points for plotting the Husimi distributions are set at IXMAX=200 and IYMAX=200. If the resolution of the Husimi distribution plot is not sufficient, increase IXMAX and IYMAX.

The intensity (i.e., the vertical axis) is shown in the log scale with the option − −vscale log.

```
$ plot2d.py dat.husimi --vscale log
```

The result is shown in Fig. 8.2.

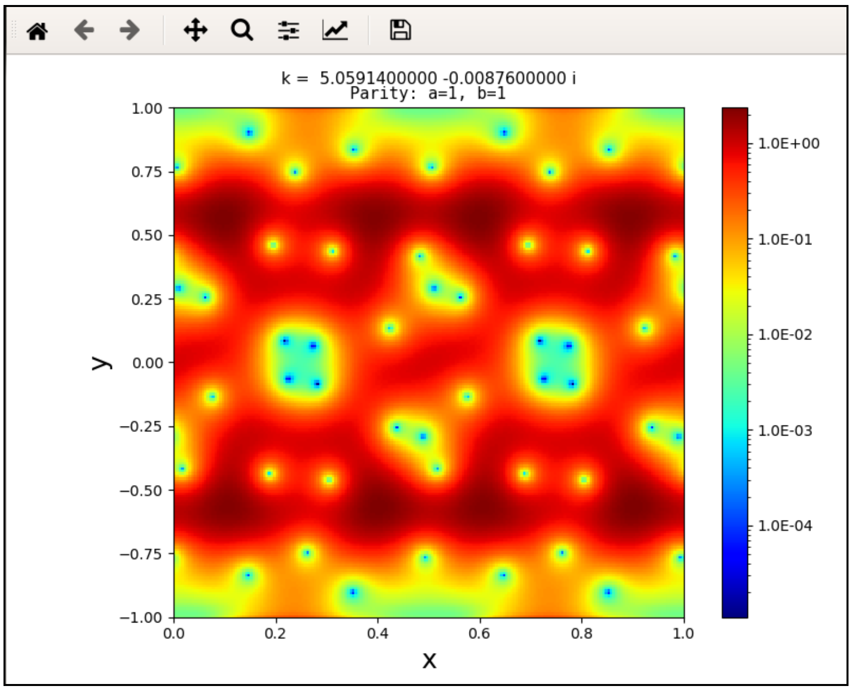


Fig. 8.2: Same as Fig. 8.1 but in log scale.

Go back to table of contents

9. Batch computation and plotting of wave functions and Husimi distributions

The python script “batch_plot.py” introduced in this section is included only in the Extension-Tools package.

The python script named batch_plot.py enables the user to automatically compute and plot wave functions and Husimi distributions for a given list of resonances. batch_plot.py is placed in $ocms/tools/ (batch_plot.py is included in the Extension-Tools package). Suppose that we have a list of resonances named “data.resonances.ee”, whose content is given as follows:

```
$ more data.resonances.ee
# Resonance data
# polarization=TM
# nin=3.3d0, nout=1.0d0
# nbe=50
# a=1
# b=1
# ---------------------------------------
# [Re k] [Im k] [det]
# ---------------------------------------
5.00512 -0.0189 0.27797916
5.05914 -0.00876 0.27591207
5.09521 -0.02455 0.28060111
```

This file contains the values of Re(k), Im(k), and determinant for three resonances for the stadium cavity with nin=3.3 and TM polarization (the detailed conditions are described in the header). For each of the resonances, batch_plot.py computes wave function and Husimi distribution data, and outputs image (png) files visualizing these data. For running batch_plot.py, the user needs to prepare a configuration file named, for example, “batch_plot_config.txt” that contains the information of executables to be used for plotting, options for these executables, and the computation conditions of the resonances. The following is an example of the configuration file corresponding to data.resonances.ee shown above:

```
$ more batch_plot_config.txt
#
# Batch_plot.py configuration file for data.resonances.ee
#

# Executable for computing wave function data
wfunc_exe=/path/to/wfunc.TM.NBE=50

# Executable for computing Husimi distribution data
husimi_exe=/path/to/husimi.TM.NBE=50
```

```
# Cavity boundary data
boundary_data=./data.cavity_boundary

# Options for plotting the wave function in normal scale
wfunc_plot_options="--boundary $boundary_data --linewidth 3"

# Options for plotting the wave function in log scale
wfunc_logplot_options="--boundary $boundary_data
--linewidth 3 --rightmargin 0.1"

# Options for plotting the Husimi distribution
husimi_plot_options=""

# Refractive index inside the cavity
nin=3.3d0

# Refractive index outside the cavity
nout=1.0d0

# Symmetry class parameter
SYM=2

# Parity indices
a=1
b=1

# Plot range for the wave function
xmin=-2.5d0
xmax=2.5d0
ymin=-1.5d0
ymax=1.5d0

# The number of data for the wave function plot
ixmax=500
iymax=300
```

batch_plot.py must be run with the two options “ – –data” and “ – –config”:

```
$ batch_plot.py --data data.resonances.ee --config batch_plot_config.txt
```

where – –data is followed by the resonance data file and – –config is followed by the configuration file. This commands creates the following three directories in this case, corresponding to the three resonances in data.resonances.ee:

```
mode.ee.k=5.00512-0.0189i/
mode.ee.k=5.05914-0.00876i/
mode.ee.k=5.09521-0.02455i/
```

In each directory, batch_plot.py outputs wave function and Husimi distribution data as well as image (png) files visualizing these data. Each of the directories contains the following files:

- Raw data:
    - dat.wfunc : Wave function data
    - dat.farfield : Far-field data
    - dat.nearfield : Near-field data
    - dat.husimi : Husimi distribution data
- Image(png) files:
    - dat.wfunc.png : Image file of the wave function data (in normal scale)
    - dat.wfunc_logscale.png : Image file of the wave function data (in log scale)
    - dat.farfield.png : Image file of the far-field data
    - dat.nearfield.png : Image file of the near-field data
    - dat.husimi.png : Image file of the Husimi distribution data
- Shell-scripts:
    - wfunc.sh : Shell-script for generating dat.wfunc, dat.farfield, and dat.nearfield
    - husimi.sh : Shell-script for generating dat.husimi

The computations of wave functions and Husimi distributions are performed in parallel. By default, batch_plot.py uses available maximum threads. The number of the threads for the parallel computation can be set by – –parallel option:

```
$ batch_plot.py --data data.resonances.ee --config batch_plot_config.txt --
parallel 4
```

In this example, 4 threads are used.

Go back to table of contents

10. Auto-detection of resonances

The python script “autofinder.py” introduced in this section is only included in the Extension-Tools package.

By the method explained in Section 6, resonances are systematically detected for a given wave number range specified by cx, cy, dwx, and dwy. However, this method requires manual routine of repeatedly running the shell-script det.sh, which could be tedious especially when many modes are detected in the search range. This manual routine can be automated by the python script “autofinder.py”, which is placed in the directory $ocms/tools/ (autofinder.py is included in the Extension-Tools package). Namely, autofinder.py automatically detects all the resonances for a given search range with a given precision.

Following the resonance search example in Section 6, let us search resonances for the stadium cavity with nin=3.3 and TM polarization for the wave number range specified by cx=5.0, cy=-0.03, dwx=0.1, and dwy=0.03 (see Section 6.1.1 for the manual search case). The corresponding options for autofinder.py are given as follows:

```
$ autofinder.py --keepall /path/to/bin/det.TM.NBE=50 nin=3.3d0 nout=1.0d0 a=1
b=1 cx=5.0d0 cy=-0.03d0 dwx=0.1d0 dwy=0.03d0 dx=0.01d0 dy=0.01d0 --epsilonx
1e-5 --epsilony 1e-5
```

Here, det.TM.NBE=50 is the executable file generated by “make det” or “make all” (see Section 4.4), and it is assumed to be placed in /path/to/bin/. The option “a=1 b=1” means that the search is for even-even parity modes.

dx and dy are the initial grid spacings, which determine the precision of the initial global search. For fully detecting the resonances without any oversight, the values of dx and dy must be set sufficiently small. Suppose that N minima of the determinant distribution are found in this search. Then, autofinder.py sets a next (second-level) search range around each of the N minima, where the center of the new search range, (cx,cy), is set to be the minimum determined up to dx and dy. The widths of the second-level search range are set to be dwx=ALPHAX * dx and dwy=ALPHAY * dy. By default, both ALPHAX and ALPHAY are set to be 1.5, but can be changed by the “ – –alphax ALPHAX” and “ – –alphay ALPHAY” options. In addition, the values of dx and dy are rescaled as MIN(BETAX * dx, BETAY * dy) → dx,dy. By default, both BETAX and BETAY are set to be 0.1, but can be changed by “ – –betax BETAX” and “ – –betay BETAY” options. One or more resonances should be detected for a second-level search, and a third-level search range is set around each of them with rescaled dwx, dwy, dx and dy. The repetition of this process increases the precision of the resonance wave numbers.

The maximum precision of the resonance search is specified by “ – –epsilonx EPSILONX” and “ – –epsilony EPSILONY” options. The resonance search routine stops when either dx < EPSILONX or dx < EPSILONY is satisfied (by default the values of EPSILONX and EPSILONY are set to be 10**(-5)). See the diagram in Section 3 of User’s Manual for the work flow diagram for resonance search and detection.

The option “−−keepall” keeps all the intermediate data files (i.e., dat.det.cx=*.cy=*.dx=*.dy=*). Without this option, all the intermediate data files are deleted at the end of the job.

For the initial global search, autofinder.py uses only a single thread. However, if multiple resonances are detected in this search, autofinder.py uses multiple threads for higher-level local searches. The number of threads, or workers, can be set by the option “−−workers WORKERS”. By default WORKERS is set to be 5.

The final output file of autofinder.py is “data.resonances.ee”, which contains the list of detected resonances (the file name changes depending on the parity). The following is the content of data.resonances.ee:

```
$ more data.resonances.ee
## ========================================
## Resonance data output by autofinder.py
## ========================================
## Command-line for autofinder.py:
## autofinder.py --keepall /path/to/bin/
## det.TM.NBE=50 nin=3.3d0 nout=1.0d0 a=1 b=1 cx=5.0d0 \
## cy=-0.03d0 dwx=0.1d0 dwy=0.03d0 dx=0.01d0 \
## dy=0.01d0 --epsilonx 1e-5 --epsilony 1e-5 \
## --sorting 0
## NOTE: missing parameters are set to default
## ========================================
## autofinder.py header ends here
## ========================================
# Resonance data
# polarization=TM
# nin=3.3d0, nout=1.0d0
# nbe=50
# a=1
# b=1
# ----------------------------------------
# [Re k] [Im k] [det]
# ----------------------------------------
5.00512 -0.0189 0.27797916
5.05914 -0.00876 0.27591207
5.09521 -0.02455 0.28060111
```

In the header, there is information regarding what options are given for autofinder.py to generate this file. Also, there is information about the physical and computational conditions for the resonance search. By default, the resonance data are sorted in the ascending order of the first column (i.e., Re(k)). The sorting column can be changed by the option “ – –sorting {0,1,2}” (0: Re(k), 1: Im(k), 2: det).

This resonance data file can be directly used for the input file for batch_plot.py.

Go back to table of contents

Appendix 1: Parameters related to the cavity symmetry

The OCMS programs take into account the following three symmetry classes labeled by the parameter SYM:

- SYM=0: The cavity does not have any mirror symmetry.
- SYM=1: The cavity is symmetric only with respect to the x-axis.
- SYM=2: The cavity is symmetric with respect to both x- and y-axis.
- SYM=4: The cavity is symmetric with respect to the diagonal x=y and y-axis.

The symmetry is used for efficiently performing the computations.

The cavity symmetry leads to the definition of parity for the eigenfunction φ(x,y) of the Helmholtz equation.

Cavity shape examples for SYM=0,1,2, and 4 are shown in Fig. A.0, where axes of mirror symmetry are superposed.

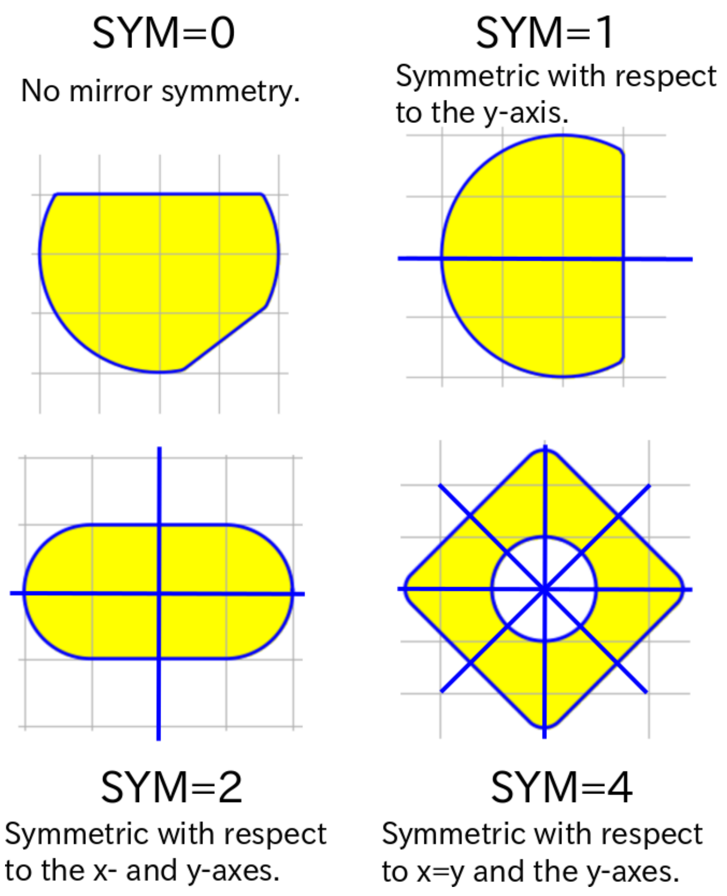

SYM=0
No mirror symmetry.
SYM=1
Symmetric with respect to the y-axis.
SYM=2
Symmetric with respect to the x- and y-axes.
SYM=4
Symmetric with respect to x=y and the y-axes.

Fig. A.0: Cavity shape examples for SYM=0,1,2, and 4 (SYM=0: Asymmetrically cut disk cavity, SYM=1: D-shaped cavity, SYM=2: Stadium cavity, SYM=4: Sinai’s cavity).

(1) SYM=4 case

For a cavity with SYM=4, the eigenfunction φ(x,y) satisfies

$$\varphi(-x,y) = a\,\varphi(x,y) \quad \text{and} \quad \varphi(y,x) = b\,\varphi(x,y),$$

with a,b ∈ {-1,+1}. When a=+1 (resp. b=+1), we say φ has *even* parity with respect to x=0 (resp. x=y). And, when a=-1 (resp. b=-1), we say φ has *odd* parity with respect to x=0 (resp. x=y). So, for SYM=4, there are four parity patterns as summarized in Table 1.

(a,b)	Parity pattern	Abbreviation
(+1,+1)	even-even	ee
(+1,-1)	even-odd	eo
(-1,+1)	odd-even	oe
(-1,-1)	odd-odd	oo

Table 1. Four parity patterns for SYM=4.

Figure A.1 illustrates four symmetry classes for a SYM=4 cavity, called Sinai's cavity. The dotted line represents even symmetry, whereas the solid line represents odd symmetry.

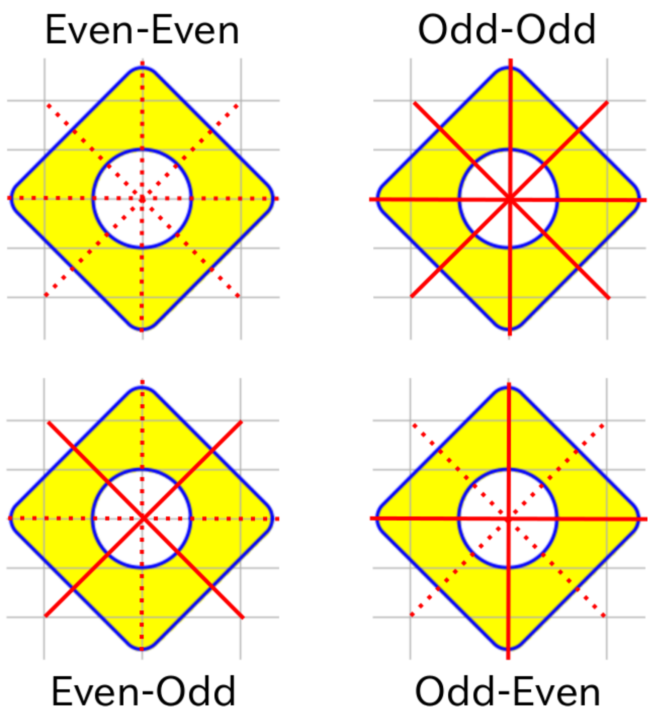


Fig. A.1: Four symmetry classes for a SYM=4 cavity.

(2) SYM=2 case

For a cavity with SYM=2, the eigenfunction φ(x,y) satisfies

$$\varphi(-x,y) = a\,\varphi(x,y) \quad \text{and} \quad \varphi(x,-y) = b\,\varphi(x,y),$$

with a,b ∈ {-1,+1}. When a=+1 (resp. b=+1), we say φ has *even* parity with respect to x=0 (resp. y=0). And, when a=-1 (resp. b=-1), we say φ has *odd* parity with respect to x=0 (resp. y=0). So, for SYM=2, there are four parity patterns as summarized in Table 2.

(a,b)	**Parity pattern**	**Abbreviation**
(+1,+1)	even-even	ee
(+1,-1)	even-odd	eo
(-1,+1)	odd-even	oe
(-1,-1)	odd-odd	oo

Table 2. Four parity patterns for SYM=2.

Figure A.2 illustrates four symmetry classes for a SYM=2 cavity, called Bunimovich's stadium cavity. The dotted line represents even symmetry, whereas the solid line represents odd symmetry.

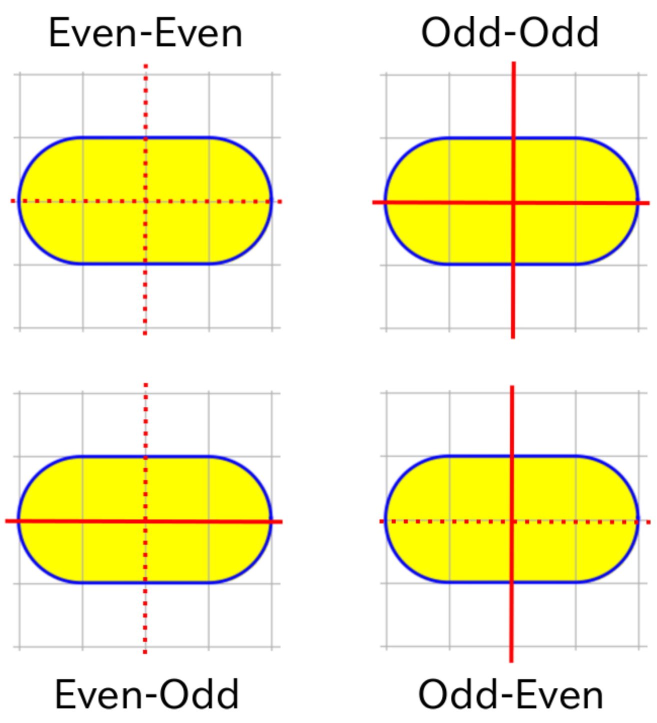


Fig. A.2: Four symmetry classes for a SYM=2 cavity.

(3) SYM=1 case

For a cavity with SYM=1, The eigenfunction φ(x,y) satisfies

$$\varphi(x,-y) = b\ \varphi(x,y),$$

with b ∈ {-1,+1}. For SYM=1, there are two parity patterns as summarized in Table 3.

b	Parity pattern	Abbreviation
+1	even	e
-1	odd	o

Table 3. Two parity patterns for SYM=1.

Figure A.3 illustrates two symmetry classes for a SYM=1 cavity, called the D-shaped cavity. The dotted line represents even symmetry, whereas the solid line represents odd symmetry.

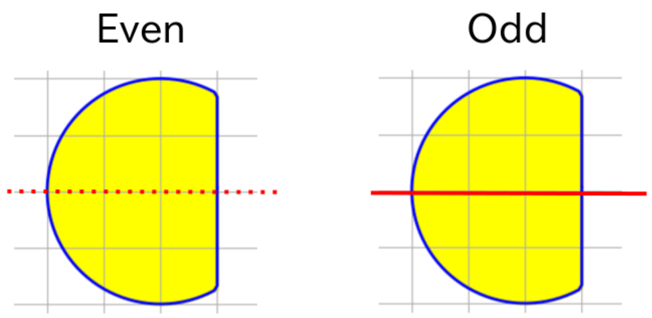


Fig. A.3: Two symmetry classes for a SYM=1 cavity.

(4) SYM=0 case

For a cavity with SYM=0, the eigenfunction does not have a parity parameter.

Go back to table of contents

Appendix 2: Summary of parameters

Symbol	Name / meaning	Type	Range	File
SYM	Cavity symmetry class	Integer	0,1,2,4	build-params.mk
CAVITY	Fortran code file name for cavity shape definition	Character	The files in $ocms/src/def_cavities	build-params.mk
SIZE_PARAM	Cavity size parameter	Double	0<SIZE_PARAM<∞	build-params.mk
PLZ	Polarization of the light field	Character	TM, TE	build-params.mk
NBE	The number of boundary elements	Integer	0<NBE<∞	build-params.mk
IXMAX,IYMAX	The number of grid points for plotting the Husimi distribution	Integer	0<IXMAX,IYMAX<∞	build-params.mk
nin	Refractive index inside the cavity	Double	0<nin<∞	det.sh, wfunc.sh, husimi.sh
nout	Refractive index outside the cavity	Double	0<nout<nin	det.sh, wfunc.sh, husimi.sh
(cx,cy)	The center of the resonance search domain in the complex wave number space	Double	0 < cx < ∞ -∞ < cy < ∞	det.sh
dwx, dwy	The half-widths of the resonance search domain in the complex wave number space	Double	0 < dwx < ∞ 0 < dwy < ∞	det.sh
dx, dy	The grid spacings for the resonance search	Double	0 < dx < dwx 0 < dy < dwy	det.sh
kx,ky	The complex wave number, k = kx + i ky, of a resonant mode	Double	0 < kx < ∞ -∞ < ky < ∞	wfunc.sh husimi.sh
xmin,xmax ymin,ymax	The xy region for outputting the wave function data	Double	-∞ < xmin < xmax< ∞ -∞ < ymin <ymax < ∞	wfunc.sh
ixmax, iymax	The number of grid points for outputting the wave function data	Integer	0 < ixmax,iymax < ∞	wfunc.sh

Go back to table of contents

Telecognix Corporation

Japan Head Office:
Yoshida Shimooji-cho 58-13
Sakyo-ku, Kyoto 606-8314
Tel: +81-75-762-4633
Fax: +81-75-762-4631

Kyoto Business Office:
612 Stork Bld. Sanjo Karasuma,
Kamanza-cho 22, Nakagyo-ku, Kyoto 604-8241
Tel: +81-75-213-3599
Fax: +81-75-213-3599

OCMS Technical Support:
Email: ocms@telecognix.com