# OCMS

## Optical Cavity Mode Solver for 2D Cavities

# User's Manual

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:**

Email: ocms@telecognix.com

# Table of contents



*** The python scripts, batch_plot.py and autofinder.py, introduced in Section 9 are only included in the Extension-Tools package.**

# 1. Introduction

The OCMS program package is used as a tool for numerically solving the eigen-problem of the two-dimensional Helmholtz equation modelling a two-dimensional optical cavity. The OCMS package enables the user to find resonances (i.e., complex wave numbers of resonant modes) for a given two-dimensional cavity, and to investigate the spatial light intensity distribution and Husimi phase space distribution of a detected resonant mode.

The OCMS package is based on the boundary element method (BEM), which has been widely used for numerically solving the Helmholtz equation. Its application to optical cavities was first demonstrated by Wiersig [1] and by Harayama et al. [2] independently. Reference [1] describes the mathematical details of the BEM for optical cavities.

In the past decade, the usefulness of the BEM has been demonstrated by many scientific studies that employ the results of the BEM for interpreting experimental data and designing new cavity shapes (for reviews, see Refs. [3-5] and references therein). The BEM is an efficient and flexible computational method with high accuracy.

It is a strong merit of the BEM that the method can be applicable to an arbitrarily shaped cavity with smooth boundaries. The OCMS package allows the user to introduce a new cavity shape by adding a code to define it. This flexibility is attractive especially for the studies of “chaotic cavities”, where the ray motion exhibits chaotic dynamics (the dependence of ray motion on the cavity shape has been studied in the mathematical field of dynamical systems and ergode theories [6]). Solving the eigen-program for a chaotic cavity is intrinsically “nonintegrable”, that is, the problem cannot be solved analytically or semi-analytically. This is in contrast to an “integrable” cavity such as a circular cavity where the resonance characteristics can be studied in a semi-analytical manner using the Bessel functions [7]. It is conjectured that any slight deformation of a circular cavity (except for elliptical deformation) results in the nonintegrability. Therefore, the nonintegrability is a rule rather than an exception.

The nonintegrability is itself an interesting property of a physical system. Whereas it is a notion defined for ray (classical) dynamics, elucidating its signature in the corresponding wave (quantum) system is a key issue of the wave (quantum) chaos theory [8]. The BEM has contributed to this research field by providing a method to systematically detect resonant modes. Such studies, combined with ray dynamical analyses, have revealed that the notions of dynamical systems theory such as periodic orbits, their stable/unstable manifolds, conditional invariant measures play an important role in precisely understanding the resonant modes of chaotic microcavities [3-5,9]. Based on this understanding, the concept of using the cavity shape as a “parameter” to control emission patterns has been examined for various cavities, even leading to a design of a unidirectional chaotic cavity laser [10]. Also, the nonintegrability was found to result in a novel emission mechanism called “chaos-assisted emission” [11], whose effects were actually observed in experiments [12,13].

As emphasized above, the flexibility to an arbitrary cavity shape and systematic resonance detection are two important features of the BEM. The OCMS package is developed so that the user can fully

utilize these features.

- References


1. J. Wiersig, “Boundary element method for resonances in dielectric microcavities”, J. Opt. A: Pure. Appl. Opt. **5**, 53 (2003).

2. T. Harayama, P. Davis, and K.S. Ikeda, “Stable oscillation of a spatially chaotic wave function in a microstadium laser”, Phys. Rev. Lett. **90**, 063901 (2003).

3. T. Harayama and S. Shinohara, “Two-dimensional microcavity lasers”, Laser Photonics Rev. **5**, 247-271 (2011).

4. S.-Y. Lee, “Optical mode properties of 2-D deformed microcavities”, in *Advances in Optical and Photonic Devices*, edited by Ki Young Kim (InTech, 2010).

5. H. Cao and J. Wiersig, “Dielectric microcavities: Model system for wave chaos and non-Hermitian physics”, Rev. Mod. Phys. **87**, 61-111 (2015).

6. N. Chernov and R. Markarian, *Chaotic billiards* (AMS, 2006).

7. J. Nöckel and A.D. Stone, “Chaotic light: a theory of asymmetric resonant cavities”, in *Optical Processes in Mirco-cavities*, edited by R.K. Chang and A.J. Campillo (World Scientific, Singapore, 1996).

8. H.-J. Stöckmann, *Quantum Chaos* (Cambridge Univ. Press, Cambridge, 1999).

9. H.G.L. Schwefel, N.B. Rex, H.E. Tureci, R.K. Chang, A.D. Stone, T. Ben-Messaoud, and J. Zyss, “Dramatic shape sensitivity of directional emission patterns from similarly deformed cylindrical polymer lasers”, J. Opt. Soc. Am. B **21**, 923-934 (2004).

10. J. Wiersig and M. Hentschel, “Combining directional light output and ultralow loss in deformed microdisks”, Phys. Rev. Lett. **100**, 033901 (2008).

11. V.A. Podolskiy and E.E. Narimanov, “Chaos-assisted tunneling in dielectric microcavities”, Opt. Lett. **30**, 474-476 (2005).

12. S. Shinohara, T. Harayama, T. Fukushima, M. Hentschel, T. Sasaki, and E.E. Narimanov, “Chaos-assisted directional light emission from microcavity lasers”, Phys. Rev. Lett. **104**, 163902 (2010).

13. J. Yang, S.-B. Lee, S. Moon, S.-Y. Lee, S.W. Kim, T.T. Anh Dao, J.-H. Lee, and K. An, “Pump-induced dynamical tunneling in a deformed microcavity laser”, Phys. Rev. Lett. **104**, 243601 (2010).


Go back to table of contents

# 2. Model and method

Here, we briefly describe the model and method related to the OCMS package. The model equation is derived from the Maxwell equations. For the derivation, see Ref. [1] for example. The boundary element method (BEM) for optical cavities is elaborated in Ref. [2].

## 2.1. Helmholtz equation

The light field of a two-dimensional optical cavity can be described by the following planar Helmholtz equation

$$\left[\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+n^2(x,y)k^2\right]\phi(x,y)=0$$

where n(x,y) describes the distribution of the refractive index and k the wave number in the vacuum. We assume that n(x,y)=nin(constant) inside the cavity, and n(x,y)=nout(constant)<nin outside the cavity. For the transverse magnetic (TM) polarization, the wave function φ(x,y) is related to the electric field as

$$\vec{E}(x,y,t)=(0,0,E_z)\ \ \text{with}\ \ E_z=\mathrm{Re}\left[\phi(x,y)e^{-ickt}\right]$$

where c is the light speed in the vacuum. At the cavity boundary, φ satisfies the following continuity conditions:

$$\phi|_{in}=\phi|_{out},\ \ \left.\frac{\partial\phi}{\partial\nu}\right|_{in}=\left.\frac{\partial\phi}{\partial\nu}\right|_{out}$$

where ∂/∂ν is the normal derivative. For the transverse electric (TE) polarization, φ is related to the magnetic field as

$$\vec{H}(x,y,t)=(0,0,H_z)\ \ \text{with}\ \ H_z=\mathrm{Re}\left[\phi(x,y)e^{-ickt}\right]$$

The continuity conditions at the cavity boundary are given by

$$\phi|_{in}=\phi|_{out},\ \ \left.\frac{1}{n_{in}^2}\frac{\partial\phi}{\partial\nu}\right|_{in}=\left.\frac{1}{n_{out}^2}\frac{\partial\phi}{\partial\nu}\right|_{out}$$

At infinity, we impose the outgoing-wave condition

$$\phi\sim\frac{f(\theta,k)}{\sqrt{r}}e^{ikr}\ \ (r\to\infty)$$

where (r,θ) are the polar coordinates. With this condition, the wave number of the resonant mode (i.e., eigen-solution) becomes complex value k=Re(k)+iIm(k) with Im(k)<0. The quality factor Q is expressed by

$$Q = \frac{\mathrm{Re}(k)}{[\mathrm{Im}(k)]^2}$$

## 2.2. Boundary element method (BEM)

The key idea of the BEM is to solve the Helmholtz equation by rewriting it as an boundary integral equation (BIE) that takes into account the boundary condition at the infinity (i.e., outgoing-wave condition). The BIE can be numerically solved by discretizing the boundary. Let us assume that the boundary is discretized into NBE elements (NBE is the number of boundary elements). The discretization of the BIE takes the following form:

$$M(k)\, x = 0$$

where x is a (2NBE)-dimensional vector, and M is a (2NBE)×(2NBE) matrix depending on k. The components of x are the values of wave function φ and its normal derivative ∂φ/∂ν along the boundary points. See Eq. (13) of Ref. [2] for the exact expressions for the matrix M and vector x.

## 2.3. Detection of the resonances

The resonances can be found by solving

$$\det(M(k)) = 0$$

where k is a complex number. This way, the problem can be reduced to a zero-point search in the complex k plane. In the OCMS package, the zero-point search is carried out in a somewhat brute force manner. Namely, we first set a search area in the complex k plane, partition it by a two-dimensional grid, evaluate the determinant for each gird point, and detect the local minimum(s) of the determinant value distribution.

## 2.4. Computation of the wave function

Once the resonant wave number k is determined, the calculation of the wave function is straightforward. By using the Green function, the wave function φ for a given point (x,y) can be expressed by an integral along the cavity boundary, where the kernel contains φ and its normal derivative ∂φ/∂ν (see Eq. (15) of Ref. [2] for the exact expression for φ). The values of φ and ∂φ/∂ν along the cavity boundary can be determined by solving M(k)x=0 for the resonant wave number k.

## 2.5. Computation of the Husimi distribution

The Husimi distribution H(s,sinθ) is used to visualize the intensity distribution in the phase space spanned by the Birkhoff coordinates (s,sinθ), where s is the normalized arclength along the cavity boundary and θ is the incident angle of a light ray. By definition, 0≤s≤1 and -1≤sinθ≤1. H(s,sinθ) represents the intensity of the light wave component coming from the inside of the cavity and hitting the cavity boundary at point s with incident angle θ. The Husimi distribution is calculated from the

wave function and its normal derivative. See Eq. (7) of Ref. [3] for the formula defining the Husimi distribution.

- References


1. H.E. Tureci, H.G.L. Schwefel, P. Jacquod, and A.D. Stone, “Modes of wave-chaotic dielectric resonators”, Prog. in Opt. **47**, 75 (2005).

2. J. Wiersig, “Boundary element method for resonances in dielectric microcavities”, J. Opt. A: Pure. Appl. Opt. **5**, 53 (2003).

3. M. Hentschel, H. Schomerus, and R. Schubert, “Husimi functions at dielectric interfaces: Inside-outside duality for optical systems and beyond”, Europhys. Lett. **62**, 636-642 (2003).


Go back to table of contents

# 3. Overview of the OCMS package

Figure 3.1 shows the directory structure of the OCMS package and brief descriptions of the directories.

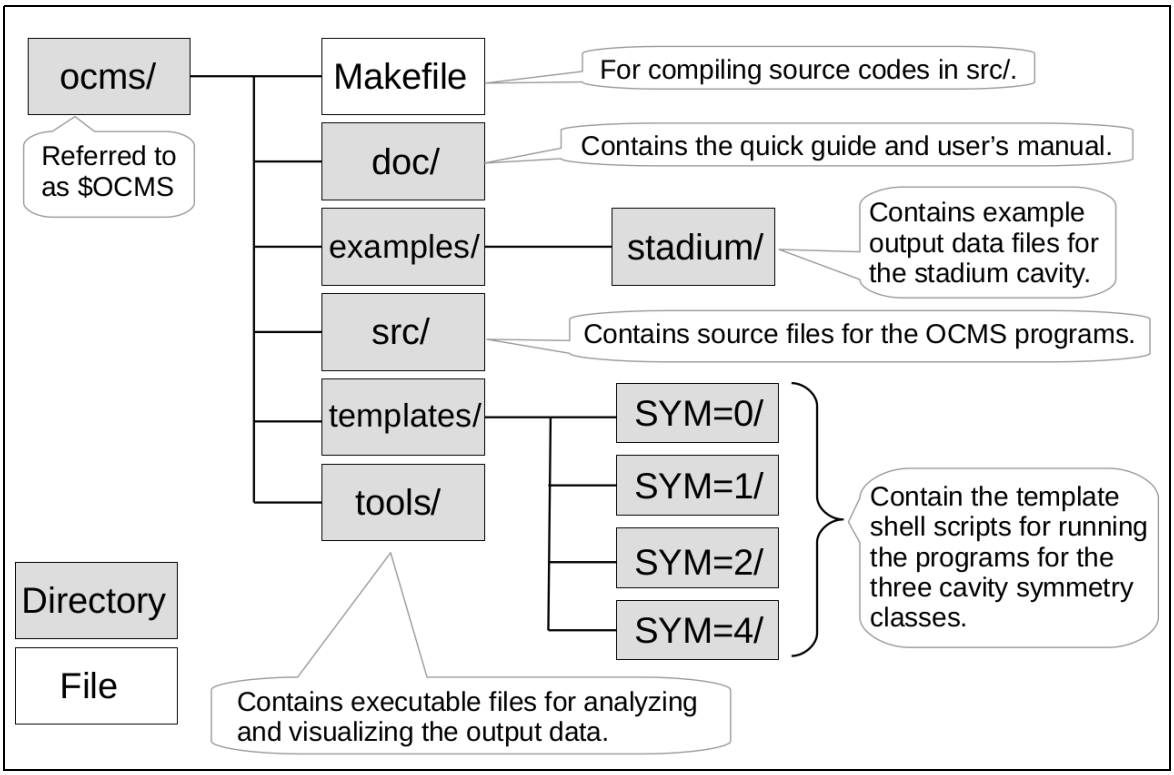



*Fig. 3.1: Directory structure of the OCMS package.*

In this user's manual, we assume that the user set OCMS as the path to the OCMS's home directory, and set PATH to include $OCMS/tools:

```
$ export OCMS=the/path/to/ocms/
$ export PATH=$PATH:$OCMS/tools
```

It is useful to set the above paths in the shell script such as .bashrc and .zschrc in your home directory.

For getting familiar with the directory structure and files involved, we recommend the user to follow the Quick Guide first.

Go back to table of contents

Figure 3.2 shows the work flow diagram for the search and detection of resonances and computation of wave functions and Husimi distributions:

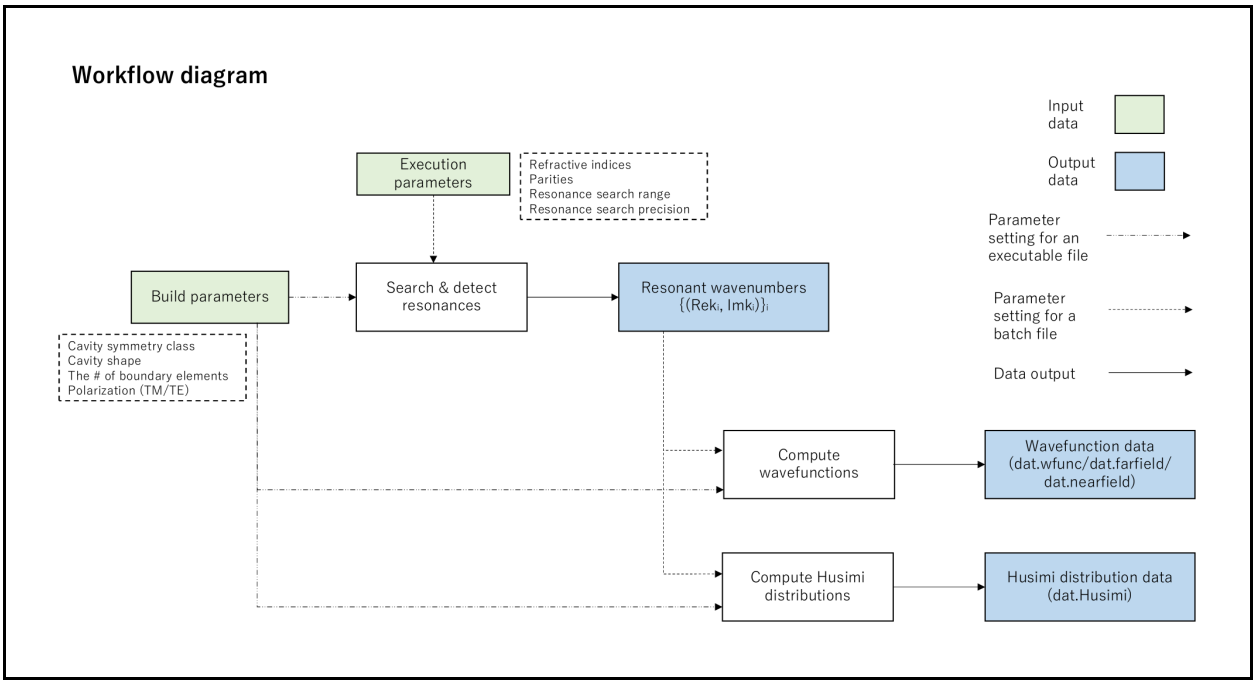



*Fig. 3.2: Workflow diagram of the OCMS package.*

Figure 3.3 shows the work flow diagram of the routine for the search and detection of resonances. dx_min and dy_min determines the precision for the resonances. The routine stops when the resonance value is determined with this precision.

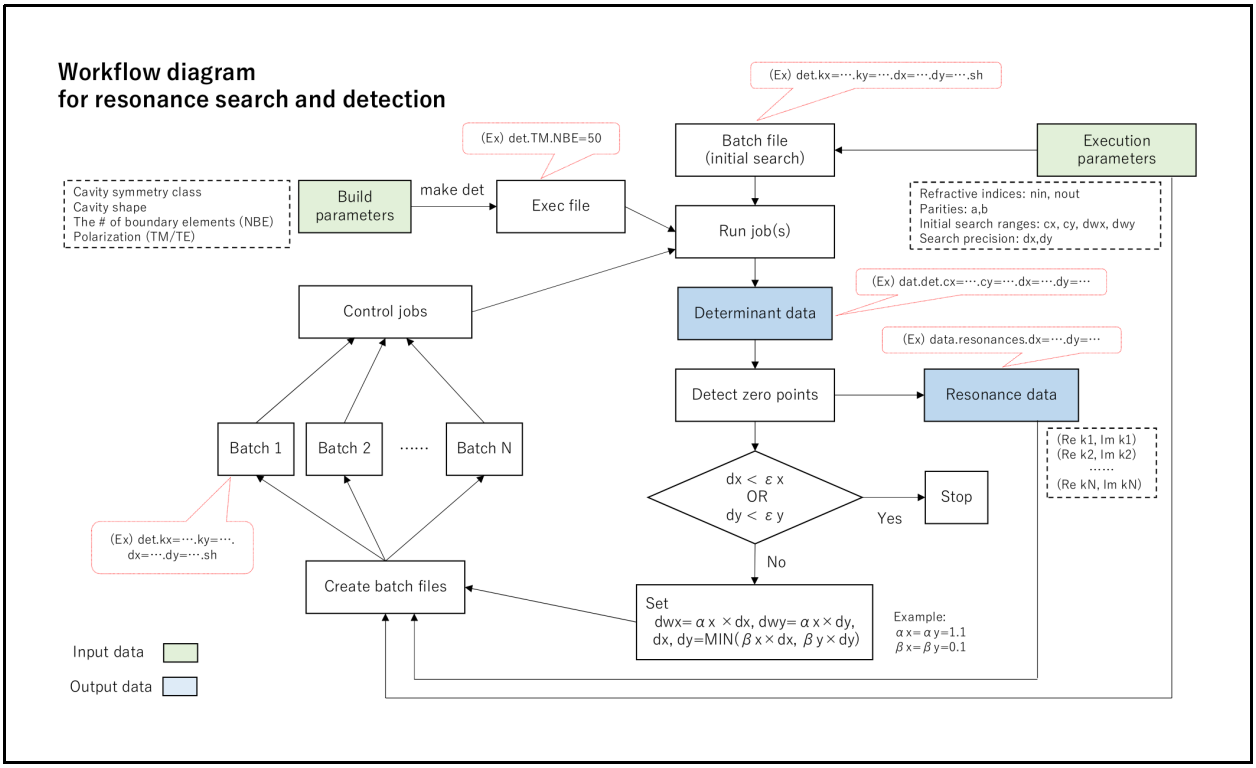



*Fig. 3.3: Workflow diagram for the detection of resonances*

Go back to table of contents

# 4. Cautions and tips

The cautions and tips when using the OCMS package are in order here:

## 4.1. Restrictions on the cavity boundary curve

BEM assumes the boundary curve to be smooth because of the divergence problem of the Green function. If your cavity shape has a singular structure like a corner, it needs to be rounded. Computations might be carried out without the rounding. However, you have to be careful about its accuracy. See Ref. [1] for the example as to the rounding of a corner.

## 4.2. Spurious modes

It is known that the boundary integral equation for a dielectric cavity contains spurious solutions. These are solutions for the interior Dirichlet problem with refractive index = 1 [1,2], 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 numerical computations, these spurious modes are found as high-quality modes (without numerical errors, they should have Im(k)=0), and you need to exclude them as spurious modes. The OCMS package offers a method to compute the wave numbers corresponding to the interior Dirichlet problem (for details, see Quick Guide, Sect. 6.3. How to distinguish spurious modes).

## 4.3. Divergence of the Greeen function at the cavity boundary

In the discretized version of the boundary integral equation, the diagonal matrix elements exhibit divergent behavior. This is due to the divergence of the Hankel function at the cavity boundary (see Sect. 3 of Ref. [1] for more details). The singularity is integrable and thus harmless in theory. However, in numerical computation, its divergent behavior needs to be treated with care. In the OCMS programs (more precisely, main.wfunc.F), the integration of the divergent term is evaluated by replacing the Green function by a divergence-free asymptotic formula. In the program, there is a parameter “BZW_HANKEL” (= boundary zone width for the computation of the Hankel function), which determines where to switch to the asymptotic formula. That is, when the distance of a given point to the cavity boundary curve is less than BZW_HANKEL, the asymptotic formula is used for the estimation of the Hankel function. The default BZW_HANKEL value is 0.005. This value should be fine for the most of the cases. However, if you greatly cares about the accuracy, we recommend the user to tune this value.

- References

1. S. Tasaki, T. Harayama, A. Shudo, “Interior Dirichlet eigenvalue problem, exterior Neumann scattering problem, and boundary element method for quantum billiards”, Phys. Rev. E, **56**, R13 (1997).

2. J. Wiersig, “Boundary element method for resonances in dielectric microcavities”, J. Opt. A: Pure. Appl. Opt. **5**, 53 (2003).

Go back to table of contents

# 5. OS and software requirements

## 5.1. Supported OS

- Ubuntu LTS version 18.04 or later

## 5.2. Software requirements

The following software tools and libraries need to be installed for using the OCMS package:

- gfortran
  Needed for compiling the Fortran codes in $OCMS/src. See Appendix 1 for how to install gfortran.

- LAPACK and BLAS
  The Fortran programs use LAPACK and BLAS for matrix manipulations. See Appendix 2 for how to install LAPACK and BLAS.

- SLATEC
  The Bessel function ZBESH() of SLATEC is used in def_hankel_zbesh.F. See Appendix 3 for how to install SLATEC.

- Python3
  Needed for the data visualization tools, minfinder.py, boundary_plot.py, plot2d.py, batch_plot.py, and autofinder.py (the last two tools are included only in the Extension-Tools package). See Appendix 4 for how to install Python 3 via anyenv.

- gnuplot
  Needed for plotting the far-field and near-field data of a resonant mode wave function. See Appendix 5 for how to install gnuplot.

Go back to table of contents

# 6. Descriptions of the files

## 6.1. Makefile

- Makefile
  - Used for compiling the Fortran codes in $OCMS/src. Compiled with the file “build-params.mk” defining the macro variables for building executable files.
  - Location: $OCMS
  - Make options:
    - make det
      creates an executable file (e.g., det.TM.NBE=50) for computing the determinant value distribution in the complex wave number space, whose zero points correspond to resonances.
    - make det_dirichlet
      creates an executable file (e.g., det_dirichlet.TM.NBE=50) for computing the determinant value distribution in the complex wave number space, whose zero points correspond to the eigenvalues for the interior Dirichlet problem.
    - make wfunc
      creates an executable file (e.g.,wfunc.TM.NBE=50) for computing the wave function data “dat.wfunc”, “dat.farfield”, and “dat.nearfield”.
    - make husimi
      creates an executable file (e.g., husimi.TM.NBE=50) for computing the Husimi distribution data “dat.husimi”.
    - make cavity
      creates an executable file (i.e., cavity) for outputting the cavity boundary and domain data “data.cavity_boundary” and “data.cavity_domain”.
    - make clean
      deletes object files and backup files (i.e., *~).
    - make cleanall
      deletes object files, backup files (i.e., *~), and executable files.
- build-params.mk
  - Used for setting macro variables for Makefile.
  - Template file Location: $OCMS/templates/SYM{0,1,2,4}

## 6.2. Shell scripts

- det.*.sh:
    - Used for running the executable file (e.g., det.TM.NBE=50, det_dirichelt.TM.NBE=50) for computing the determinant value distribution.
    - Template file location: $OCMS/templates/SYM{0,1,2,4}
- wfunc.*.sh
    - Used for running the executable file (e.g., wfunc.TM.NBE=50) for the computation of the wave function data.
    - Template file location: $OCMS/templates/SYM{0,1,2,4}
- husimi.*.sh
    - Used for running the executable file (e.g. husimi.TM.NBE=50) for the computation of the Husimi distribution.
    - Template file location: $OCMS/templates/SYM{0,1,2,4}

## 6.3. Main programs

The following main programs are located in $OCMS/src.

- main.det.F
  Main program for computing and outputting the determinant value distribution.
- main.det_dirichlet.F
  Main program for computing and outputting the determinant value distribution for the interior Dirichlet problem.
- main.wfunc.F
  Main program for computing and outputting the wave function data.
- main.husimi.F
  Main program for computing and outputting the Husimi distribution data.
- main.cavity.F
  Main program for outputting the cavity boundary and domain data.
- main.estimateNBE.F90
  Main program for returning the number of boundary elements NBE for given refractive index nin, wave number k, and the ratio between the half wavelength inside the cavity and the maximum length of the boundary element.

- main.estimateRatio.F90
  Main program for returning the ratio between the half wavelength inside the cavity and the maximum length of the boundary element for given refractive index nin, wave number k, and the number of boundary element NBE.

## 6.4. Subprograms

The following subprograms except def_cavity_SYM{0,1,2,4}_[cavity shape].F are in $OCMS/src. def_cavity_SYM{0,1,2,4}_[cavity shape].F are in $OCMS/src/def_cavities.

- def_green_func.F
  Greeen functions are defined.
- def_hankel_zbesh.F
  Hankel functions are defined using zbesh() of SLATEC.
- def_matrix.F
  Matrix for the boundary integral equation is defined.
- def_cavity_SYM_{0,1,2,4}_[cavity shape].F
  The boundary and domain of the cavity shape are defined.

Go back to table of contents

## 6.5. Executable files

The following executable files are created by Makefile to the same directory as the Makefile is located.

- det.{TM,TE}.NBE=[INTEGER] (e.g., det.TM.NBE=50)
  - Computes the determinant value distribution.
  - Output file: dat.det.cx=…cy=…dx=…dy=…
  - Created by Makefile with “make det”.
  - Executed by det.*.sh.
- det_Dirichlet.{TM,TE}.NBE=[INTEGER] (e.g., det.TM.NBE=50)
  - Computes the determinant value distribution for the interior Dirichlet problem.
  - Output file: dat.det.cx=…cy=…dx=…dy=…
  - Created by Makefile with “make det”.
  - Executed by det.*.sh.

- wfunc.{TM,TE}.NBE=[INTEGER] (e.g., wfunc.TM.NBE=50)
    - Computes the wave function of a resonant mode.
    - Output files:
        - dat.wfunc: data for the real and imaginary values of the wave function.
        - dat.farfield: far-field intensity distribution.
        - dat.nearfield: near-field intensity distribution.
    - Created by Makefile with “make wfunc”.
    - Executed by wfunc.*.sh.
- husimi.{TM,TE}.NBE=[INTEGER] (e.g., husimi.TM.NBE=50)
    - Computes the Husimi distribution.
    - Output file: dat.husimi
    - Created by Makefile with “make husimi”.
    - Executed by husimi.sh.
- cavity_data
    - Outputs the following two data files:
        - data.cavity_boundary: data for defining the cavity boundary.
        - data.cavity_domain: data for defining the cavity domain.
    - Created by Makefile with “make cavity”.
    - Executed by ./cavity_data
- estimateNBE
    - Returns the minimum value of NBE (the number of boundary elements) for which the ratio between the half wavelength inside the cavity and the maximum length of the boundary elements exceeds a given value.
    - Created by Makefile with “make estimateNBE”.
    - Executed by ./estimateNBE [nin] [k] [ratio]

- estimateRatio
    - Returns the value of the ratio between the half wavelength inside the cavity and the maximum length of the boundary elements.
    - Created by Makefile with “make estimateRatio”.
    - Executed by ./estimateRatio [nin] [k] [NBE]

Go back to table of contents

## 6.6. Output data files

- dat.det.cx=…cy=…dx=…dy=…
  - Creator: det.{TM,TE}.NBE=[INTRGER]
  - Content: The values of the determinant in the complex wave number space. For avoiding underflow, the determinant values are scaled as
    $$(\det\ M)' = (\det\ M)^{10^{-\beta}}$$
    with β>0, where β is automatically tuned in the program.
  - minfinder.py is used for finding the local minimum(s) of this distribution.
- dat.wfunc
  - Creator: wfunc.{TM,TE}.NBE=[INTEGER]
  - Content: The real and imaginary part of the wave function φ(x,y).
  - plot2d.py is used for plotting the intensity distribution |φ(x,y)|^2.
  - See also Section 10.1 for the contents of this file.
- dat.farfield
  - Creator: wfunc.{TM,TE}.NBE=[INTEGER]
  - Content: The far-field intensity distribution, |f(θ,k)|^2, i.e., the dependence of the absolute square of the scattering amplitude on the polar angle.
  - farfield.plt is used for plotting the far-field intensity distribution.
- dat.nearfield
  - Creator: wfunc.{TM,TE}.NBE=[INTEGER]
  - Content: The near-field pattern, i.e., the field intensity distribution along the cavity boundary.
  - nearfield.plt is used for plotting the near-field intensity distribution.
- dat.husimi
  - Creator: husimi.{TM,TE}.NBE=[INTRGER]
  - Content: The Husimi distribution
  - plot2d.py is used for plotting the Husimi distribution.
  - See also Section 10.1 for the contents of this file.

- data.cavity_boundary
  - Creator: cavity_data
  - Content: This file contains a data frame with the following column labels:

```
# [n] [x(n)] [y(n)] [nx(n)] [ny(n)] [ds(n)] [kappa(n)]

n: the index for the boundary element (1<=n<=NBE)
x(n): the x coordinate for the center of the n-th boundary element
y(n): the y coordinate for the center of the n-th boundary element
nx(n): the x coordinate of the normal vector at (x(n),y(n))
ny(n): the y coordinate of the normal vector at (x(n),y(n))
ds(n): the length of the n-th boundary element
kappa(n): the cavity boundary curvature at (x(n),y(n))
```

  - boundary_plot.py is used for visualizing the content of this file.
  - See also Section 10.1 for the contents of this file.
- data.cavity_domain
  - Creator: cavity_data
  - Content: This file contains a set of the points (x,y) inside the cavity.
  - boundary_plot.py is used for visualizing the content of this file.
- data.resonances.{ee,eo,oe,oo}, data.resonances.{e,o}, data.resonances
  - Creator: autofinder.py
  - Content: List of resonance data consisting of Re(k), Im(k), and the determinant value.
  - This file is used as the input file for batch_plot.py.

## 6.7. Scripts

The following python scripts (*.py) and gnuplot scripts (*.plt) are located in $OCMS/tools.

- minfinder.py
  - Used to find local minimum(s) of the determinant value distribution generated by det.{TM,TE}.NBE=[INTEGER].
- plot2d.py
  - Used to plot two-dimensional data in dat.wfunc and dat.husimi.

- boundary_plot.py
    - Used to visualize the boundary and domain data in data.cavity_boundary and data.cavity_domain.
- farfield.plt
    - Used to plot the farfield intensity distribution in dat.farfield.
- nearfield.plt
    - Used to plot the nearfield intensity distribution in dat.nearfield.
- batch_plot.py
    - Used for computing and plotting wave function and Husimi distribution data for each mode listed in the resonance data file.
    - batch_plot.py is included in the Extension-Tools package.
- autofinder.py
    - Used for detecting all the resonances in a given wave number range with a given precision.
    - autofinder.py is included in the Extension-Tools package.

## 6.8. Example files

- $OCMS/examples/stadium/
  contains example data files for the stadium cavity (TM-polarization, nin=3.3, nout=1.0, k≈5, even-even polarization), and build-params.mk and shell-scripts for generated the data files.

## 6.9. Template files

The template file for build parameter setting for Makefile (i.e., build-params.mk) and template shell-scripts for running the executable files (i.e., det.sh, wfunc.sh, and husimi.sh) are located in $OCMS/template. The template files are different for different symmetry classes. The sub-directories SYM0, SYM1, SYM2, and SYM4, respectively contain templates suitable for SYM=0 (cavities without mirror symmetry), SYM=1 (cavities that are symmetric only with respect to the x axis), SYM=2 (cavities that are symmetric with respect to both x and y axes), and SYM=4 (cavities that have $C_{4v}$ symmetry, i.e., symmetric with respect to the diagonal x=y and the y axis).

Go back to table of contents

# 7. Parameters

## 7.1. Overview

Parameters are set in the following files:

- build-params.mk
- det.sh
- wfunc.sh
- husimi.sh

build-params.mk is for parameter setting for building executable files, while the shell-scripts are for parameter settings for the executable files.

Go back to table of contents

## 7.2. Parameters for Makefile

The following parameters are set in build-params.mk:

- SYM
  - Type: Integer
  - Range: 0,1,2,4
  - Default: –
  - Description: Specifies the symmetry of the cavity:
    - SYM=0: The cavity does not have any mirror symmetry.
    - SYM=1: The cavity is symmetric only with respect to the y-axis.
    - SYM=2: The cavity is symmetric with respect to both x- and y-axes.
    - SYM=4: The cavity is symmetric with respect to the diagonal x=y and the y-axis.
  - Remark: SYM has to be consistent with the cavity shape specified by CAVITY.

- CAVITY
  - Type: String
  - Range: –
  - Default: –
  - Description: Specifies the Fortran code to define the cavity shape:
    - CAVITY=def_cavity_SYM0_Sinai.F : Sinai's cavity
    - CAVITY=def_cavity_SYM0_asym_cut_disk.F: Asymmetrically cut disk
    - CAVITY=def_cavity_SYM0_asym_limacon.F: Asymmetric limacon cavity
    - CAVITY=def_cavity_SYM0_rounded_triangle.F: Rounded triangular cavity
    - CAVITY=def_cavity_SYM1_D-shape.F: D-shaped cavity
    - CAVITY=def_cavity_SYM1_annular.F: Annular cavity
    - CAVITY=def_cavity_SYM1_cardioid.F: Cardioid cavity
    - CAVITY=def_cavity_SYM2_D2_deformed_circle.F: Deformed circular cavity with $D_2$ symmetry
    - CAVITY=def_cavity_SYM2_Sinai.F : Sinai's cavity
    - CAVITY=def_cavity_SYM2_ellipse.F: Elliptic cavity
    - CAVITY=def_cavity_SYM2_flattened_quadrupole.F: Flattened quadrupole cavity
    - CAVITY=def_cavity_SYM2_stadium.F: Stadium cavity
    - CAVITY=def_cavity_SYM4_Sinai.F : Sinai's cavity
  - Remark: The program will terminate with an error message, if SYM contradicts CAVITY.
- NBE
  - Type: Integer
  - Range: 0<NBE
  - Default: –
  - Description: The number of the boundary elements.
  - Remark: NBE is determined so that the maximal length of the boundary elements is

sufficiently smaller than the wavelength inside the cavity. The program “estimateNBE” helps finding an appropriate NBE value.

- PLZ
  - Type: String
  - Range: TM or TE
  - Default: –
  - Description: Polarization of the light fields
- IXMAX, IYMAX
  - Type: Integer
  - Range: 0 < IXMAX, IYMAX
  - Default: IXMAX=200, IYMAX=200
  - Description: These parameters are used in main.husimi.F. IXMAX and IYMAX represent the number of data points in the x direction and that in the y direction, respectively.

Go back to table of contents

## 7.3. Parameters for det.sh

- nin
  - Type: Float
  - Range: 0 < nout < nin
  - Default: –
  - Description: Refractive index inside the cavity.
- nout
  - Type: Float
  - Range: 0 < nout < nin
  - Default: 1.0
  - Description: Refractive index outside the cavity.

- a
  - Type: Integer
  - Range: -1 or 1
  - Default: –
  - Description: Specifies the parity of the mode with respect to the y-axis (see Section 7.6 for the definition of the parity).
    - a=1: Even parity
    - a=-1: Odd parity
  - Remark: When SYM=0 or SYM=1, this parameter is absent.
- b
  - Type: Integer
  - Range: -1 or 1
  - Default: –
  - Description: Specifies the parity of the mode with respect to the x-axis for SYM=1 and SYM=2, and to the diagonal x=y for SYM=4 (see Section 7.6 for the definition of the parity).
    - b=1: Even parity
    - b=-1: Odd parity
  - Remark: When SYM=0, this parameter is absent.
- cx, cy
  - Type: Float
  - Range: 0.0 < cx, cy
  - Default: –
  - Description: The point (cx,cy) specifies the center of the resonance search domain in the complex wave number space.
- dwx, dwy
  - Type: Float
  - Range: 0.0 < dwx, dwy

- Default: –
- Description: dwx and dwy specify the half-widths of the resonance search domain in the kx and ky directions, respectively.
- Remark: dwx must be chosen to satisfy cx-dwx > 0.

- dx,dy
  - Type: Float
  - Range: 0.0 < dx, dy
  - Default: –
  - Description: dx and dy specify the grid spacing in the kx and ky directions, respectively.
  - Remark: Too large dx and dy might lead to the failure of resonance detection.

Figure 7.1 shows the resonance search domain specified by cx, cy, dwx, dwy.

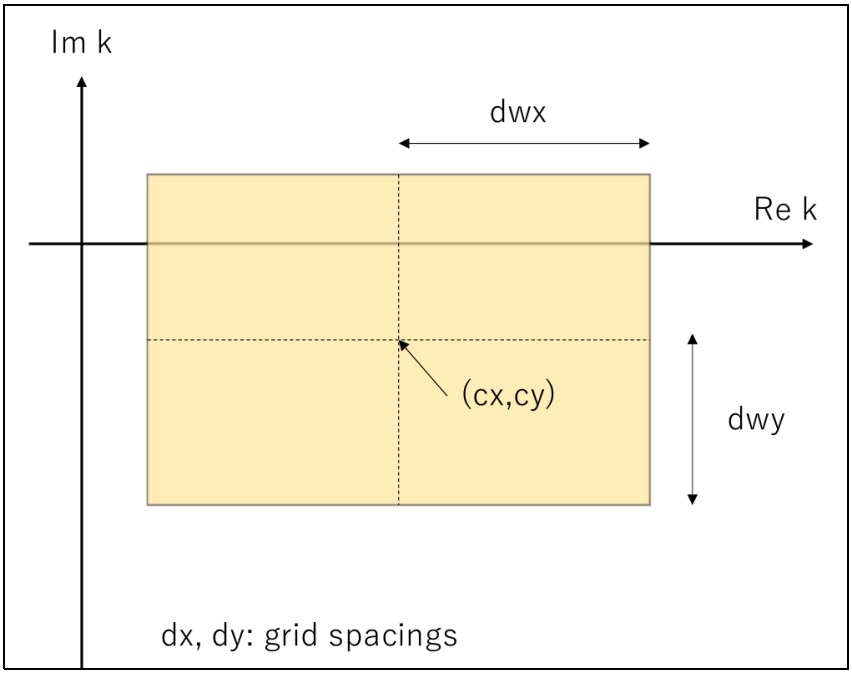



*Fig. 7.1: Two-dimensional domain for resonance search in the complex wave number plane.*

Go back to table of contents

## 7.4. Parameters for wfunc.sh

- nin
  - Type: Float
  - Range: 0 < nout < nin
  - Default: –
  - Description: Refractive index inside the cavity.
- nout
  - Type: Float
  - Range: 0 < nout < nin
  - Default: 1.0
  - Description: Refractive index outside the cavity.
- a
  - Type: Integer
  - Range: -1 or 1
  - Default: –
  - Description: Specifies the parity of the mode with respect to the y-axis (see Section 7.6 for the definition of the parity).
    - a=1: Even parity
    - a=-1: Odd parity
  - Remark: When SYM=0 or SYM=1, this parameter is absent.
- b
  - Type: Integer
  - Range: -1,1
  - Default: –
  - Description: Specifies the parity of the mode with respect to the x-axis for SYM=1 and SYM=2, and to the diagonal x=y for SYM=4 (see Section 7.6 for the definition of the parity).

        - b=1: Even parity
        - b=-1: Odd parity
    - Remark: When SYM=0, this parameter is absent.

- kx, ky
    - Type: Float
    - Range: 0 < kx, ky
    - Default: –
    - Description: kx and ky represent the real and imaginary parts of the complex wave number of a resonant mode, respectively.

- xmin, xmax
    - Type: Float
    - Range: xmin < xmax
    - Default: –
    - Description: xmin and xmax specify the x-range of the area for computing the field intensity distribution of a resonant mode.

- ymin, ymax
    - Type: Float
    - Range: ymin < ymax
    - Default: –
    - Description: ymin and ymax specify the y-range of the area for computing the field intensity distribution of a resonant mode.

- ixmax, iymax
    - Type: Integer
    - Range: 0 < ixmax, iymax
    - Default: ixmax=300, iymax=300
    - Description: ixmax and iymax are the numbers of grid points in the x- and y-directions, respectively.

## 7.5. Parameters for husimi.sh

- nin
  - Type: Float
  - Range: 0 < nout < nin
  - Default: –
  - Description: Refractive index inside the cavity.
- nout
  - Type: Float
  - Range: 0 < nout < nin
  - Default: 1.0
  - Description: Refractive index outside the cavity.
- a
  - Type: Integer
  - Range: -1 or 1
  - Default: –
  - Description: Specifies the parity of the mode with respect to the y-axis (see Section 7.6 for the definition of the parity).
    - a=1: Even parity
    - a=-1: Odd parity
  - Remark: When SYM=0 or SYM=1, this parameter is absent.
- b
  - Type: Integer
  - Range: -1 or 1
  - Default: –
  - Description: Specifies the parity of the mode with respect to the x-axis for SYM=1 and SYM=2, and to the diagonal x=y for SYM=4 (see Section 7.6 for the definition of the parity).

        - b=1: Even parity
        - b=-1: Odd parity
    - Remark: When SYM=0, this parameter is absent.

- kx, ky
    - Type: Float
    - Range: 0 < kx, ky
    - Default: –
    - Description: kx and ky represent the real and imaginary parts of the complex wave number of a resonant mode, respectively.

Go back to table of contents

## 7.6. Summary of the symmetry-related parameters

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

- SYM=0: The cavity has no 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, that is, the cavity has $C_{2v}$ symmetry.
- SYM=4: The cavity is symmetric with respect to the diagonal x=y and the y-axis, that the cavity has $C_{4v}$ symmetry.

The symmetry is used for efficiently performing the computations. The cavity symmetry leads to the definition of parity for the eigenfunction $\varphi(x,y)$ of the Helmholtz equation.

Cavity shape examples for SYM=0,1,2, and 4 are shown in Fig. 7.1, where axes of mirror symmetry are superposed.

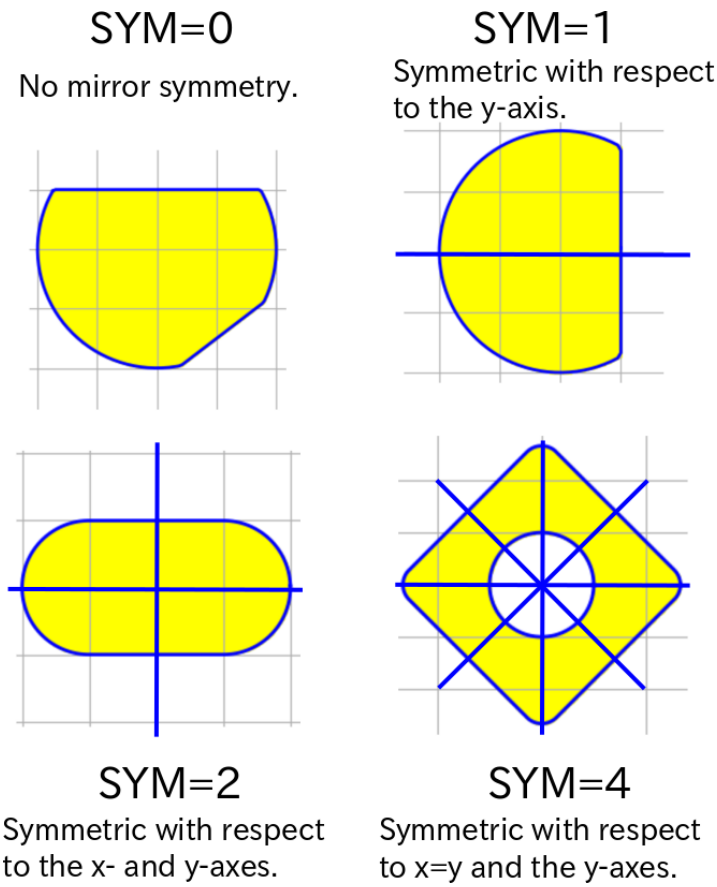



*Fig. 7.1: Cavity shape examples for SYM=0,1,2, and 4 (SYM=0: Asymmetrically cut disk cavity, SYM=1: D-shaped cavity, SYM=2: Stadium cavity, SYM=4: Sinai's cavity).*

## (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 7.2: 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.

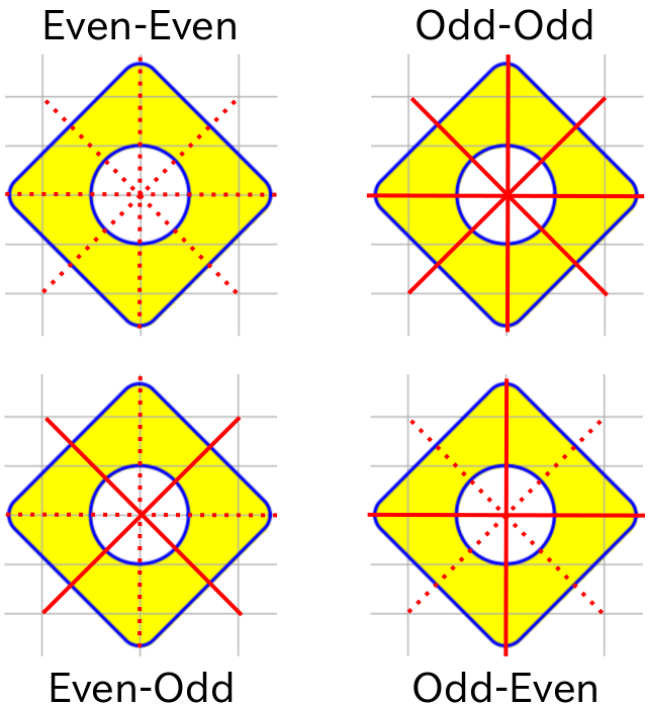



*Fig. 7.2: Four symmetry classes for a SYM=4 cavity.*

## (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 7.3 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.

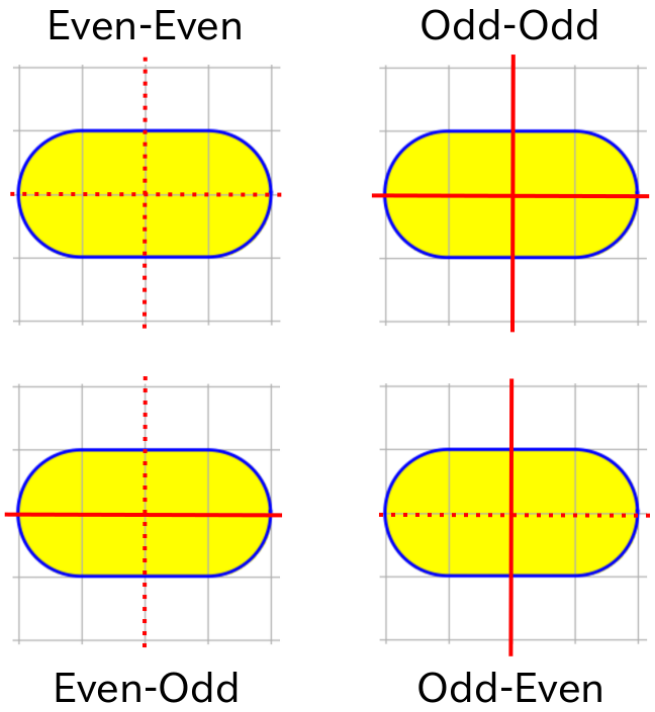



*Fig. 7.3: Four symmetry classes for a SYM=2 cavity.*

## (3) SYM=1 case

For a cavity with SYM=1, The eigenfunction $\varphi(x,y)$ satisfies

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

with $b \in \{-1,+1\}$. 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 7.4 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.

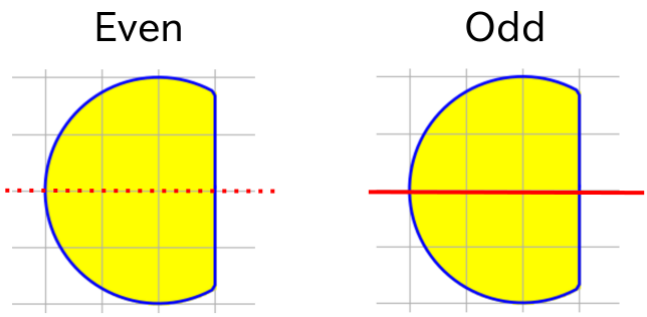



*Fig. 7.4: Two symmetry classes for a SYM=1 cavity.*

## (4) SYM=0 case

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

Go back to table of contents

# 8. Visualization tools

## 8.1. minfinder.py

This python script is used for detecting the local minimum(s) of the determinant distribution. This command needs to be fed with a determinant value data file (e.g., dat.det.cx=…cy=…dx=…dy=…).

### 8.1.1. Usage

```
$ minfinder.py --help
usage: minfinder.py [-h] [--nodisplay] [--savefig] [--nozeropoints]
[--colormap COLORMAP] [--annotate] [--labelsize LABELSIZE]
[--ticksize TICKSIZE] [--textsize TEXTSIZE]
[--textcolor TEXTCOLOR]
filename

Tool for finding local minimums of a determinant value distribution in the
complex wave number space.

positional arguments:
filename determinant value data file

optional arguments:
-h, --help show this help message and exit
--nodisplay don't display plot figure
--savefig save figure image as png
--nozeropoints don't output local minimums found
--colormap COLORMAP set colormap(default is 'viridis_r')
--annotate show annotations of local minimums found
--labelsize LABELSIZE
set font size of axes label (default is 18)
--ticksize TICKSIZE set font size of tick labels (default is 14)
--textsize TEXTSIZE set font size of text (default is 9)
--textcolor TEXTCOLOR
set text color(default is 'black')
```

Go back to table of contents

### 8.1.2. Detecting the minimum(s) and plotting the determinant value distribution

Assume that the user is in the directory $OCMS/example/stadium/resonance_search.

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

This command outputs a three-column list of detected minimums to the standard output as follows:

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

The first, second, and third columns represent Re(k), Im(k), and scaled determinant values, respectively. Also, this command opens a window showing the determinant value distribution in the complex wave number space. The window is closed by pushing the “q” button.

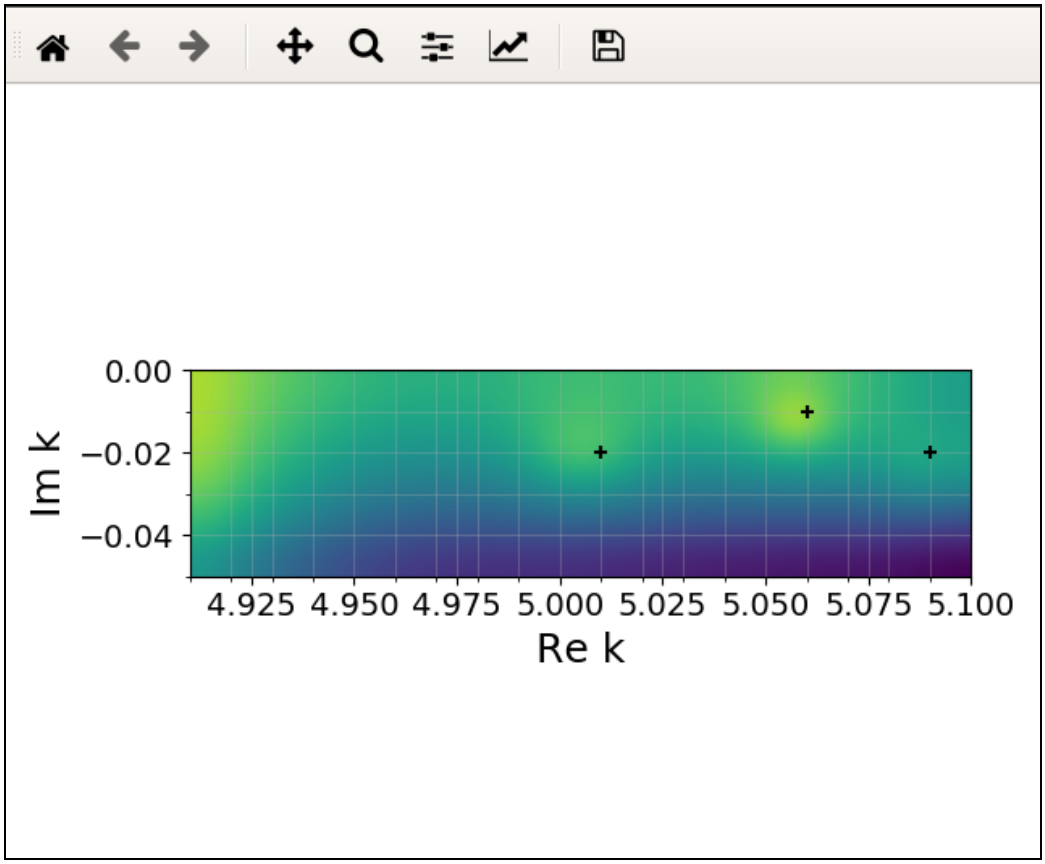



*Fig. 8.1: Distribution of determinant values in the complex k plane, plotted by minfinder.py*.

Go back to table of contents

### 8.1.3. Top bar buttons of the Matplotlib interface

The functions of the top bar buttons are described in Fig. 8.2.

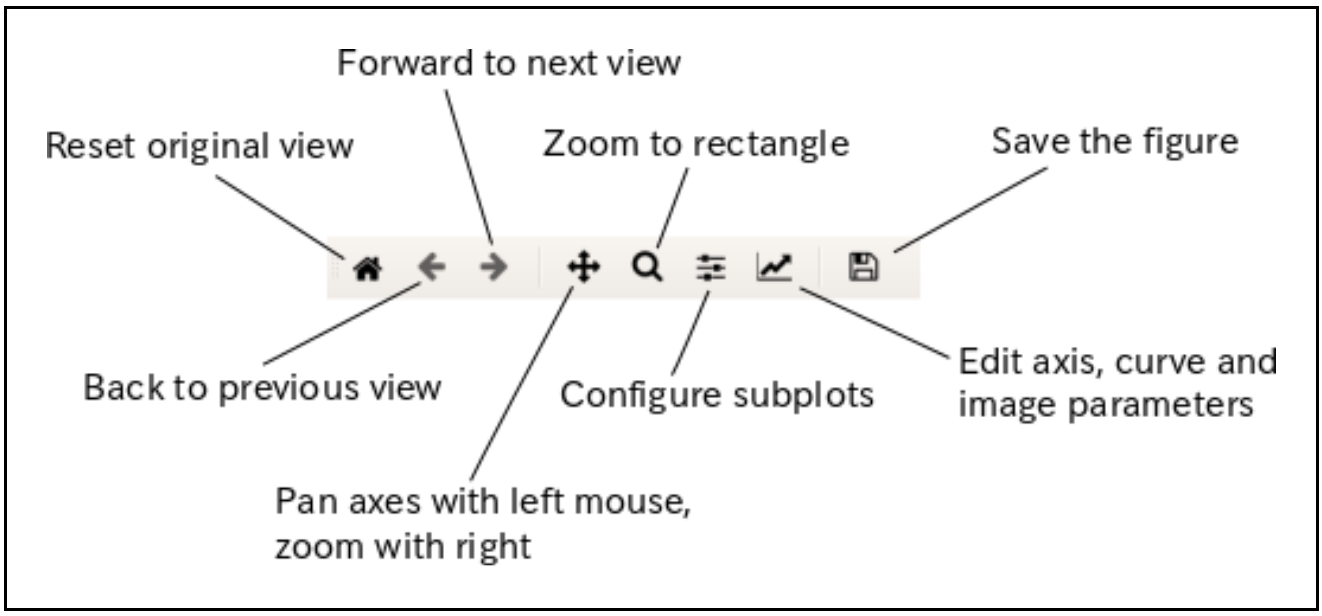



*Fig. 8.2: Functions of the top bar buttons.*

Go back to table of contents

### 8.1.4. Superposing the position coordinates of the local minimum(s).

Use – –annotate option:

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

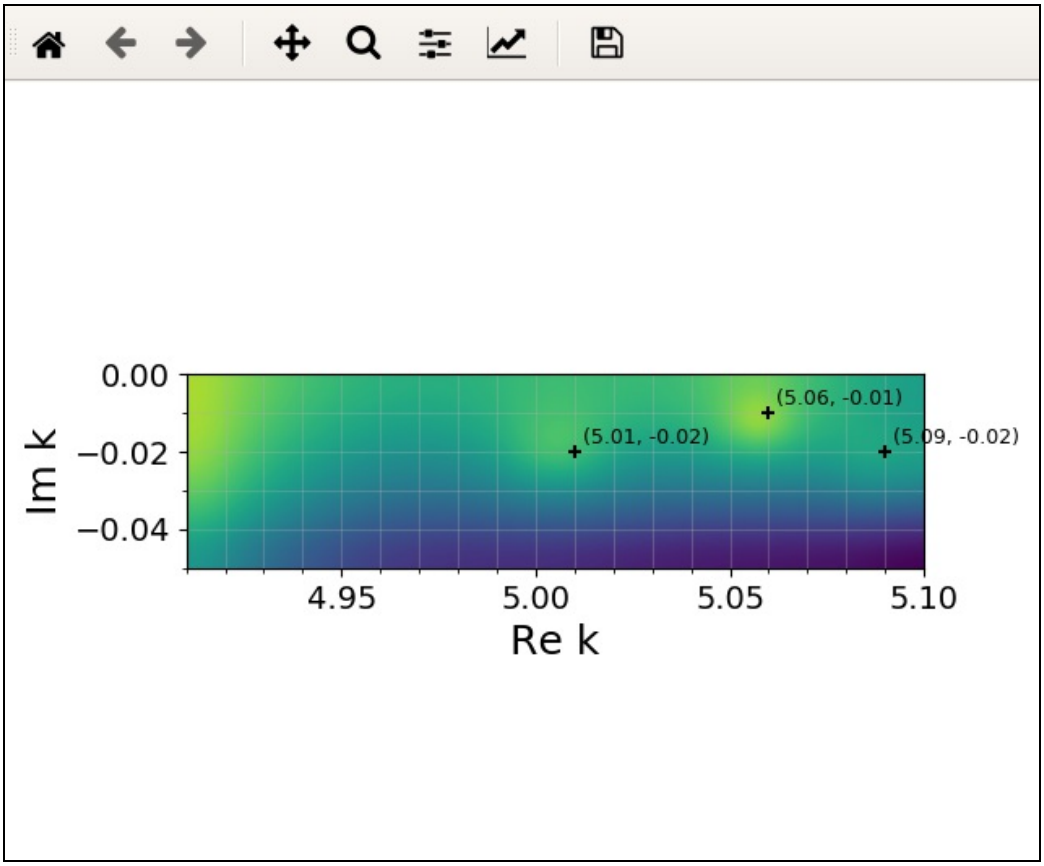



*Fig. 8.3: Distribution of determinant values in the complex k plane, plotted by minfinder.py with – –annotate option.*

Go back to table of contents

### 8.1.5. Save the image as a png file

Use both – –savefig and – –nodisplay options:

```
$ minfinder.py dat.det.cx\=5.0d0.cy\=-0.03d0.dx\=0.01d0.dy\=0.01d0 --annotate
--savefig --nodisplay
5.095 -0.025 0.29399456
5.06 -0.01 0.29310035
5.005 -0.02 0.29368768
save image file: dat.det.cx=5.0d0.cy=-0.03d0.dx=0.01d0.dy=0.01d0.png ....
```

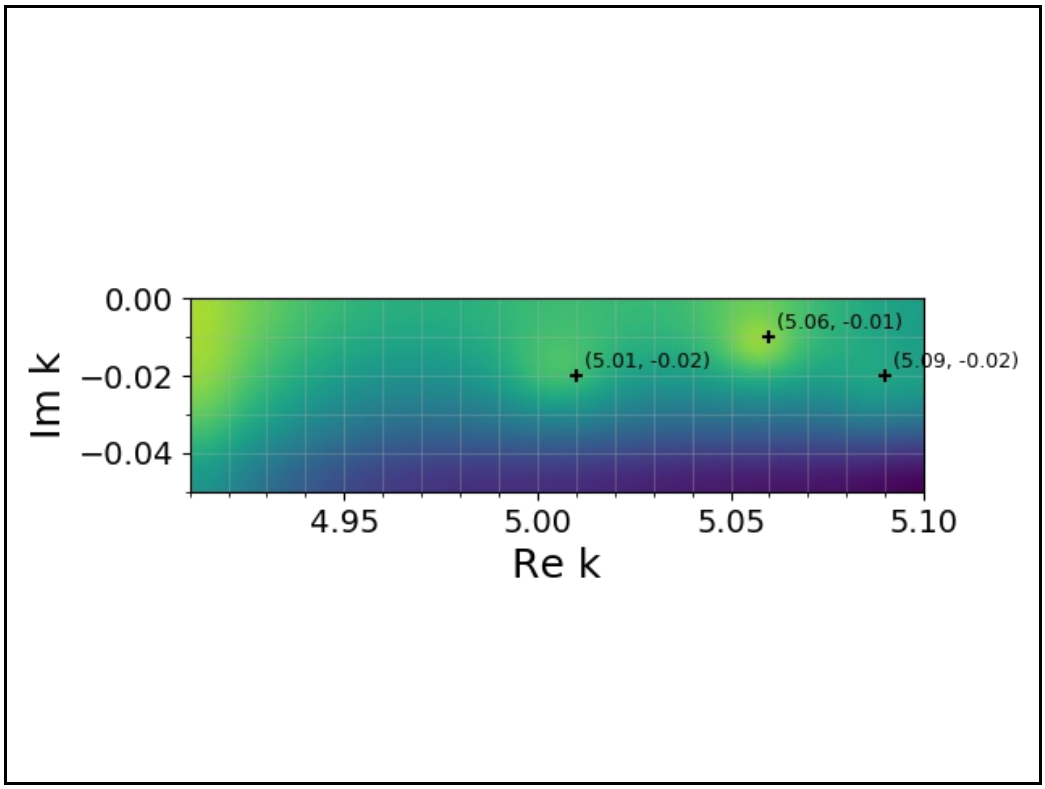



*Fig. 8.4: Saved image.*

Go back to table of contents

## 8.2. plot2d.py

This python script is used for visualizing two-dimensional data such as dat.wfunc and dat.husimi.

### 8.2.1. Usage

```
$ plot2d.py --help
usage: plot2d.py [-h] [--savefig] [--colormap COLORMAP]
[--colorbarformat COLORBARFORMAT] [--nocolorbar]
[--nocolorbarticks] [--rightmargin [0.0 to 1.0]]
[--vscale [linear/log]] [--vmin VMIN] [--vmax VMAX]
[--boundary BOUNDARY] [--color COLOR] [--linewidth LINEWIDTH]
[--labelsize LABELSIZE] [--ticksize TICKSIZE]
[--textsize TEXTSIZE] [--dataformat [auto/real/complex/mtv]]
[--debug]
filename

Tool for plotting a wave function and a Husimi distribution.

positional arguments:
filename wave function or Husimi distribution data file

optional arguments:
-h, --help show this help message and exit
--savefig save figure image as png
--colormap COLORMAP set colormap(default is 'jet')
--colorbarformat COLORBARFORMAT
set format of colorbar ticks
--nocolorbar don't display colorbar
--nocolorbarticks don't display ticks of colorbar
--rightmargin [0.0 to 1.0]
set ratio of right-margin
--vscale [linear/log]
set the v-axis scale (default is 'linear')
--vmin VMIN
--vmax VMAX
--boundary BOUNDARY draw cavity boundary
--color COLOR set color of boundary line (default is 'white')
--linewidth LINEWIDTH
set width of boundary line (default is 2)
--labelsize LABELSIZE
set font size of axes label (default is 18)
--ticksize TICKSIZE set font size of tick labels (default is 14)
--textsize TEXTSIZE set font size of text (default is 11)
--dataformat [auto/real/complex/mtv]
set format of data file (default is 'auto')
--debug output debug log
```

### 8.2.2. Plotting the wave function data

Assume that the user is in the directory $OCMS/example/stadium/mode.ee.kx=5.05914-0.00876i.

```
$ plot2d.py dat.wfunc
```

This command opens a window showing the intensity distribution of the wave function as shown in Fig. 8.5.

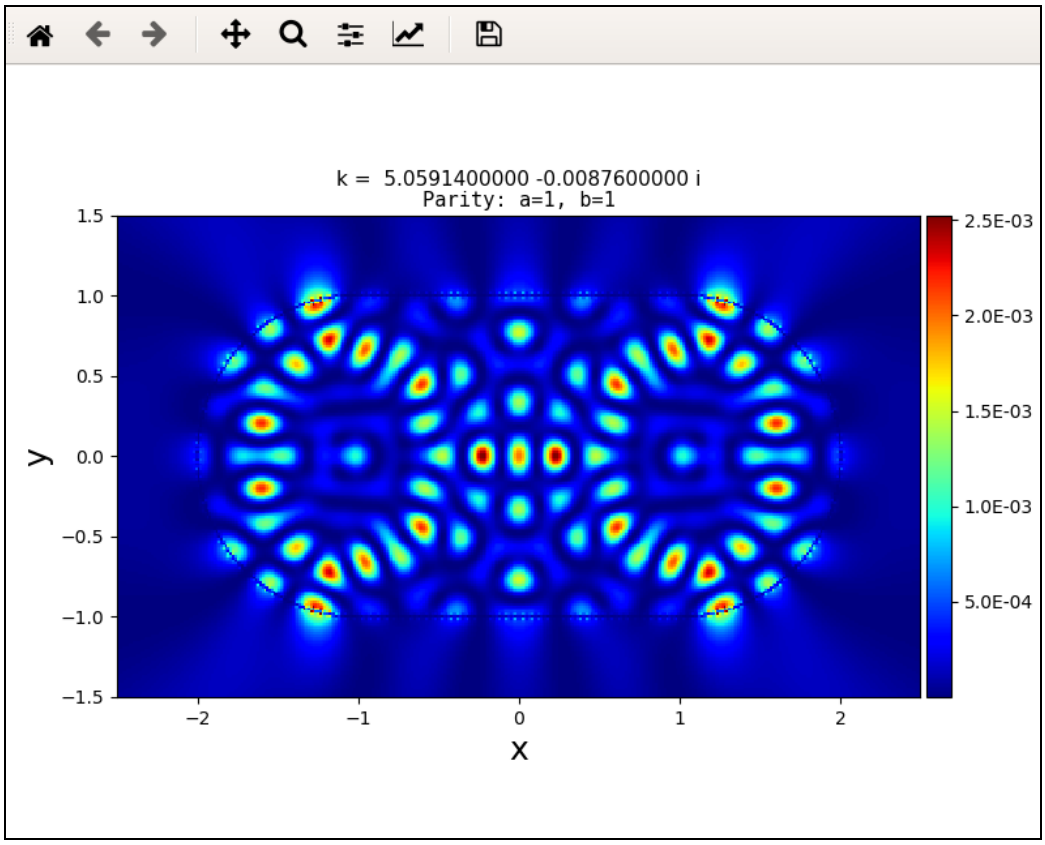



*Fig. 8.5: Intensity distribution of a wave function plotted by plot2d.py.*

Go back to table of contents

### 8.2.3. Plotting with a boundary curve

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

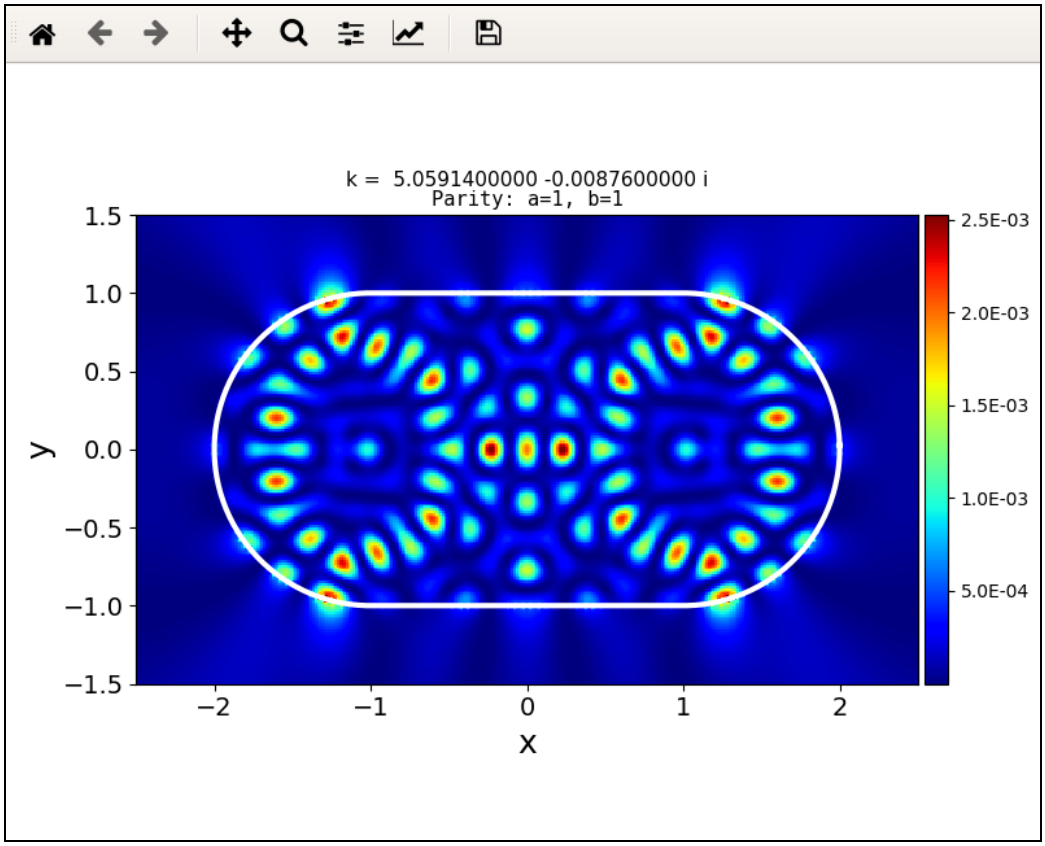



*Fig. 8.6: Intensity distribution of a wave function plotted by plot2d.py with --boundary option.*

The width of the boundary curve can be changed by --linewidth [REAL NUMBER] option (default linewidth is 2.0).

The color of the boundary curve can be changed by --color [COLOR] option (default color is 'white').

Go back to table of contents

### 8.2.4. Plotting in log scale

Use `--vscale log` option:

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

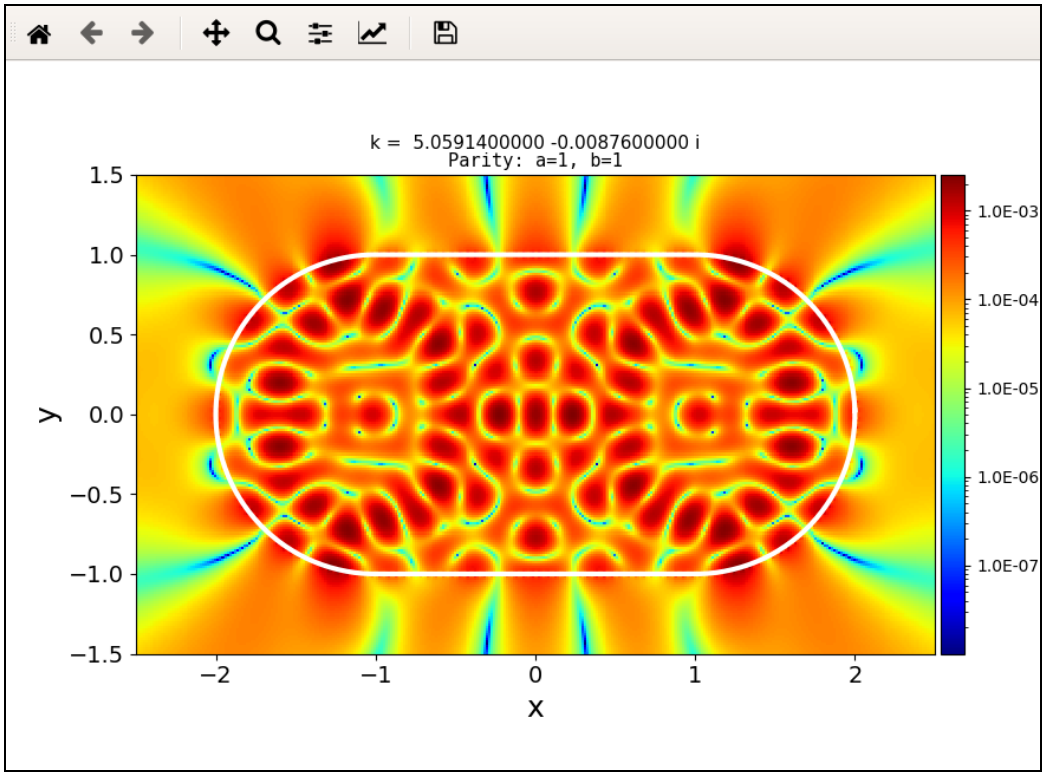



*Fig. 8.7: Intensity distribution of a wave function in log scale.*

Go back to table of contents

### 8.2.5. Saving the image as a png file

Use – –savefig option:

```
$ plot2d.py dat.wfunc --boundary ../data.cavity_boundary --linewidth 3 --
savefig
save image file: dat.wfunc.png ....
```

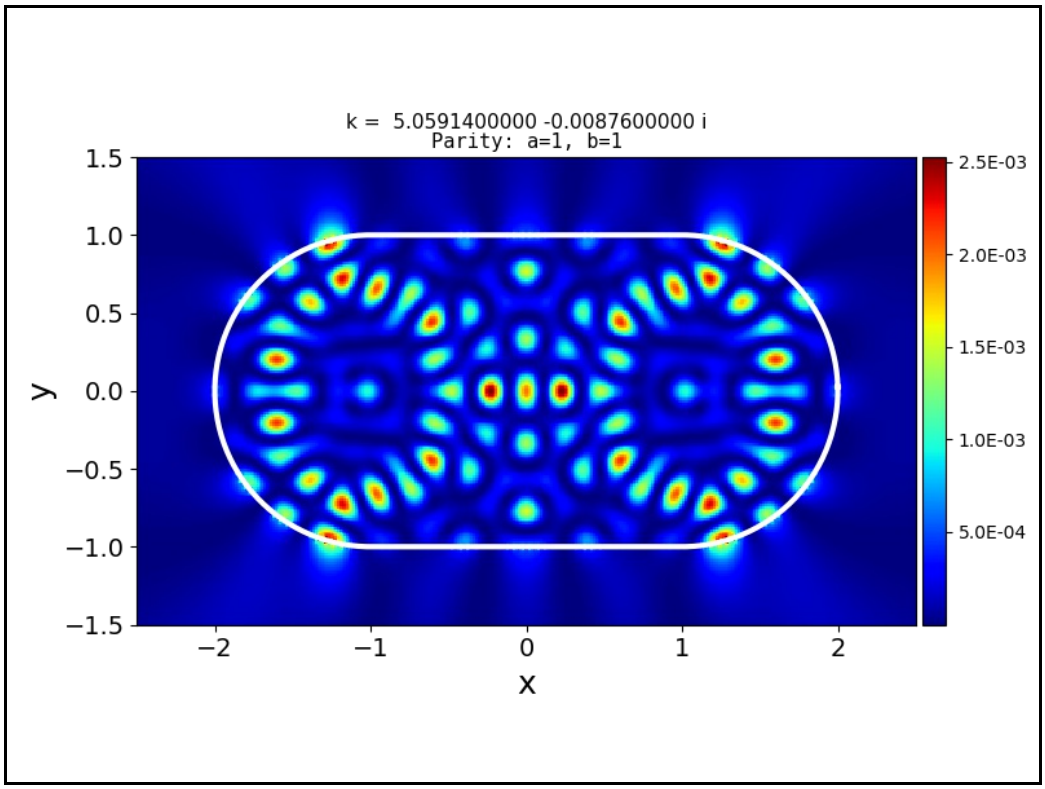



*Fig. 8.8: Saved image.*

Go back to table of contents

### 8.2.6. Plotting the Husimi distribution data

Assume that the user is in the directory $OCMS/example/stadium/mode.ee.kx=5.05914-0.00876i.

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

This command opens a window showing the Husimi distribution as shown in Fig. 8.9.

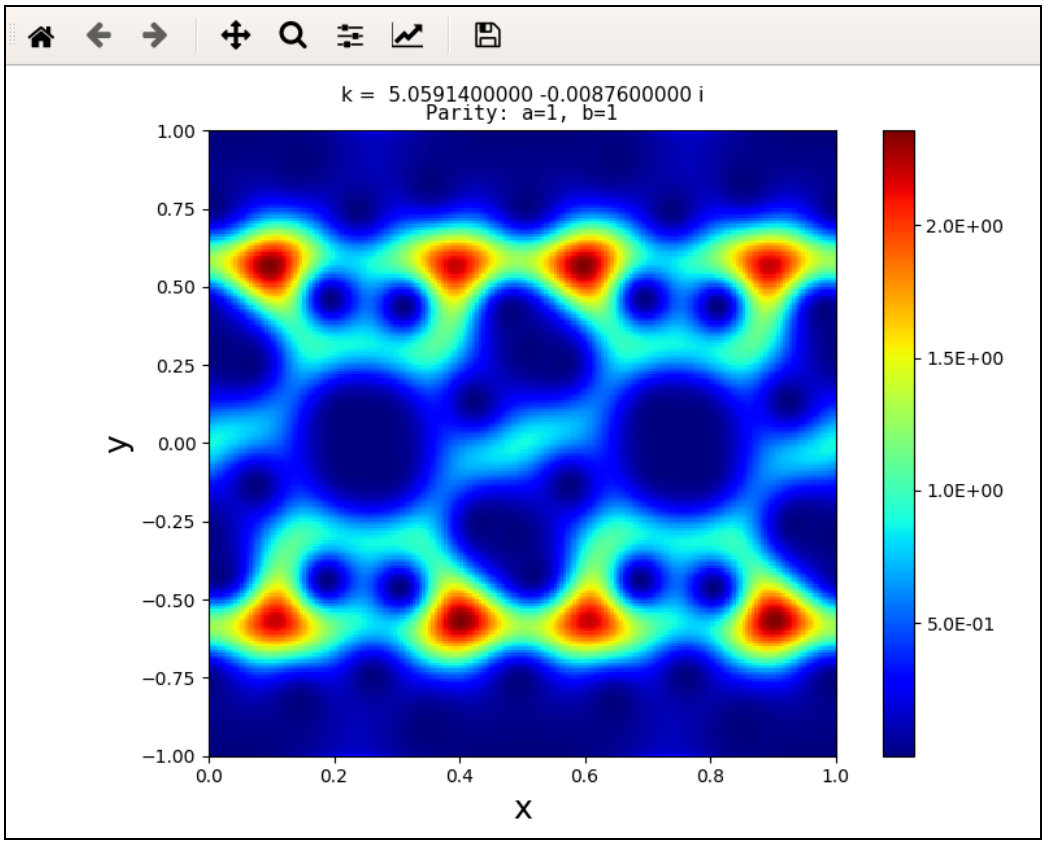



*Fig. 8.9: Husimi distribution of a wave function plotted by plot2d.py.*

Go back to table of contents

### 8.2.7. Plotting in log scale

Use `--vscale log` option:

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

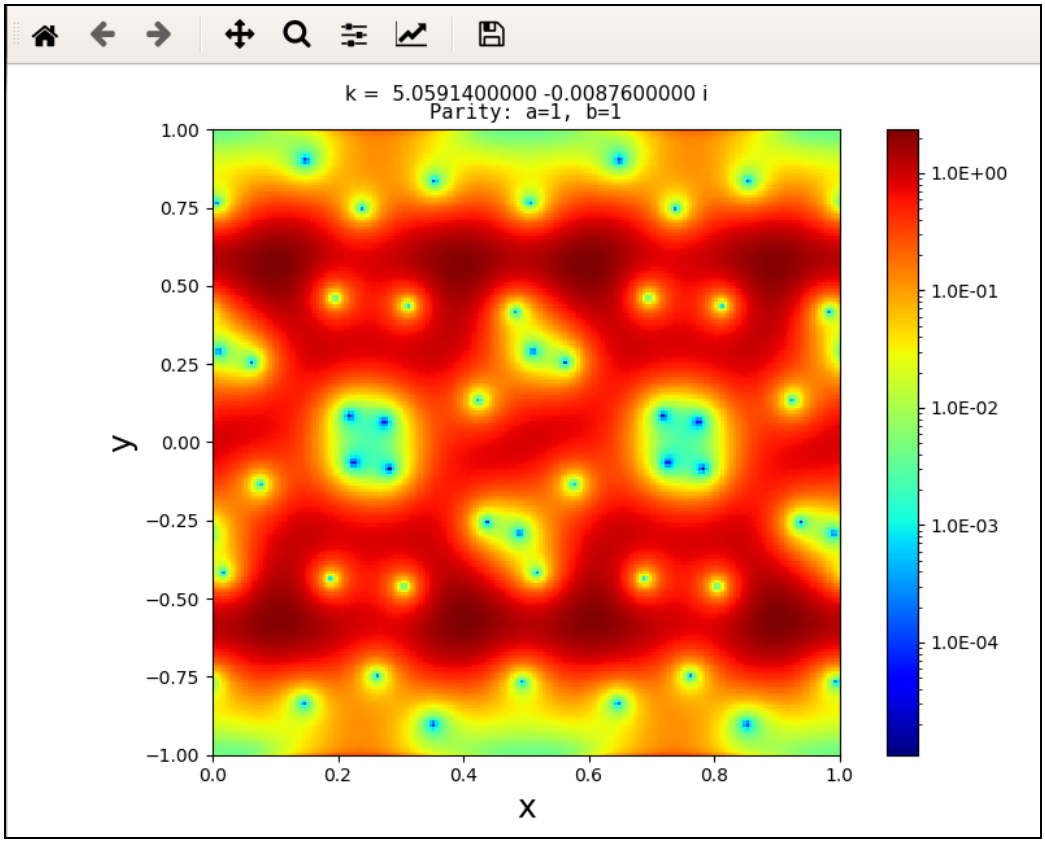



*Fig. 8.10: Husimi distribution of a wave function in log scale.*

Go back to table of contents

### 8.2.8. Saving the image as a png file

Use – –savefig option:

```
$ plot2d.py dat.husimi --savefig
save image file: dat.husimi.png ....
```

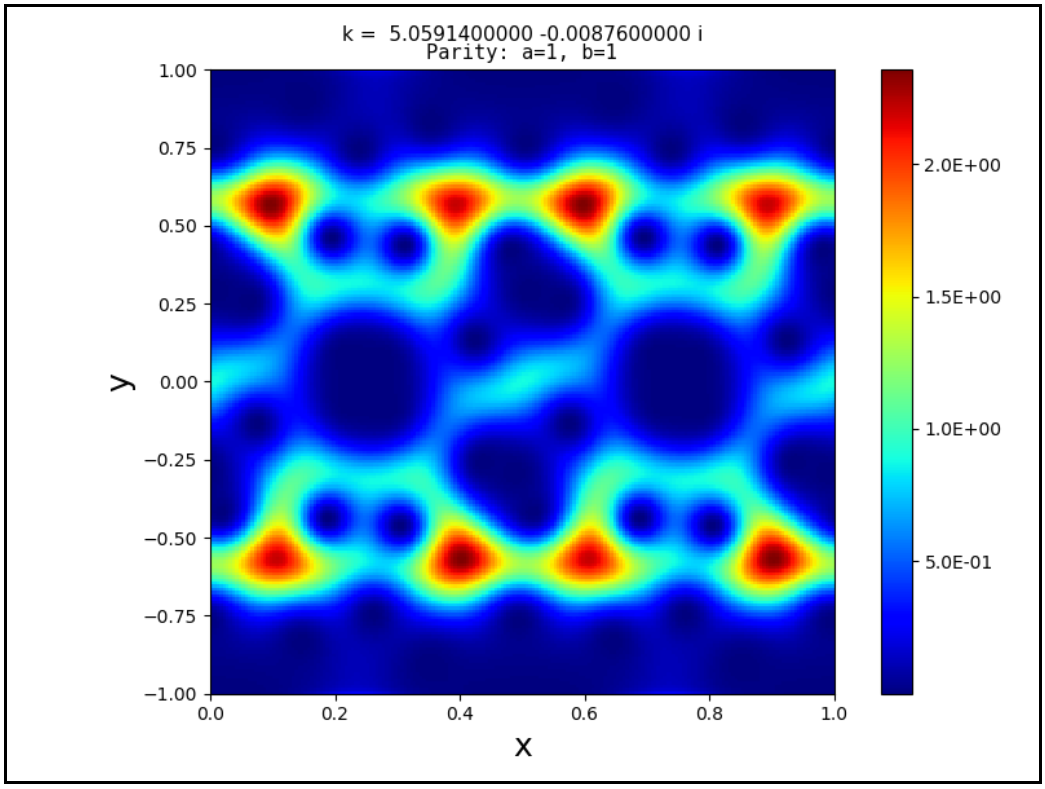



*Fig. 8.11: Saved image.*

Go back to table of contents

## 8.3. boundary_plot.py

This python script can be for debugging the cavity shape definition code (i.e., def_cavity_SYM{0,1,2,4}_[cavity shape].F). It reads the two cavity data files, data.cavity_boundary and data.cavity_area, and visualize information on the cavity domain, boundary, normal vectors, and the dependence of the curvatures, and the lengths of the boundary elements on the boundary element index n ($1 \leq n \leq NBE$).

For the plot of normal vectors (see the top right panel of the figure in the next page), their length is automatically rescaled. The actual length is given by RATIO*max(WIDTH,HEIGHT), where WIDTH and HEIGHT are the maximal width and height of the cavity, respectively, and RATIO is set to be 0.1 by default.

### 8.3.1. Usage

```
$ boundary_plot.py --help
usage: boundary_plot.py [-h] --boundary BOUNDARY [--domain DOMAIN] [--debug]

Tool for visualizing the information of a cavity shape and its boundary.

optional arguments:
-h, --help show this help message and exit
--boundary BOUNDARY load cavity boundary data
--domain DOMAIN load cavity domain data
--debug output debug log
```

Go back to table of contents

### 8.3.2. Visualizing the information on the cavity shape and boundary

Assume that the user is in the directory $OCMS/example/stadium.

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

This command opens a window as shown in Fig. 8.12.

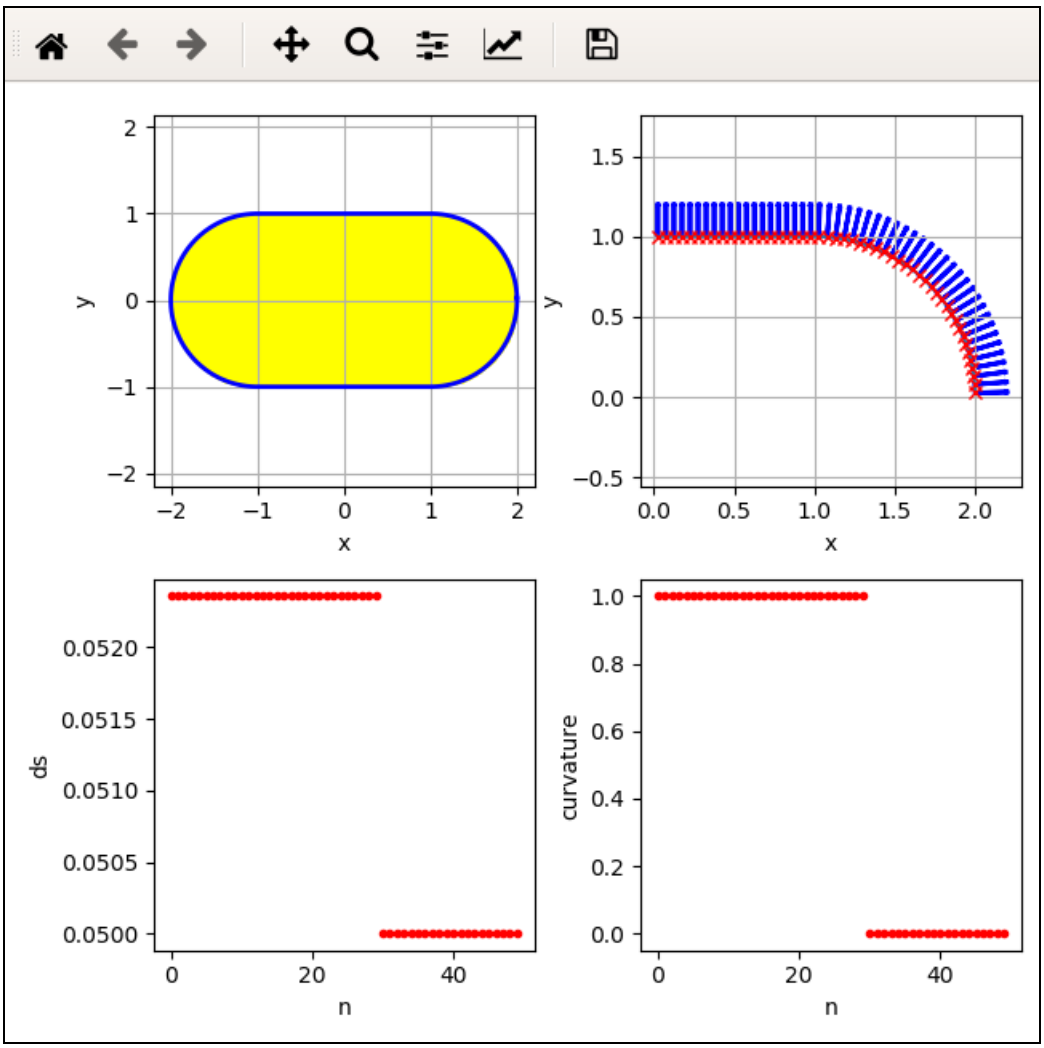



*Fig. 8.12: The top left panel shows the cavity domain defined by data.cavity_domain (yellow region) and the cavity boundary curve defined by data.cavity_boundary (blue curve), the top right panel the cavity boundary points (red crosses) with normal vectors (blue lines), the bottom left panel the dependence of the length of a boundary element on the cavity boundary element index n, and the bottom right panel the dependence of the curvature on the cavity boundary element index n.*

Go back to table of contents

## 8.4. farfield.plt

This is a gnuplot script for plotting the far-field data “dat.farfield”, and is executed as follows:

```
$ farfield.plt
```

This command creates an image file “dat.farfield.png” as shown in Fig. 8.13.

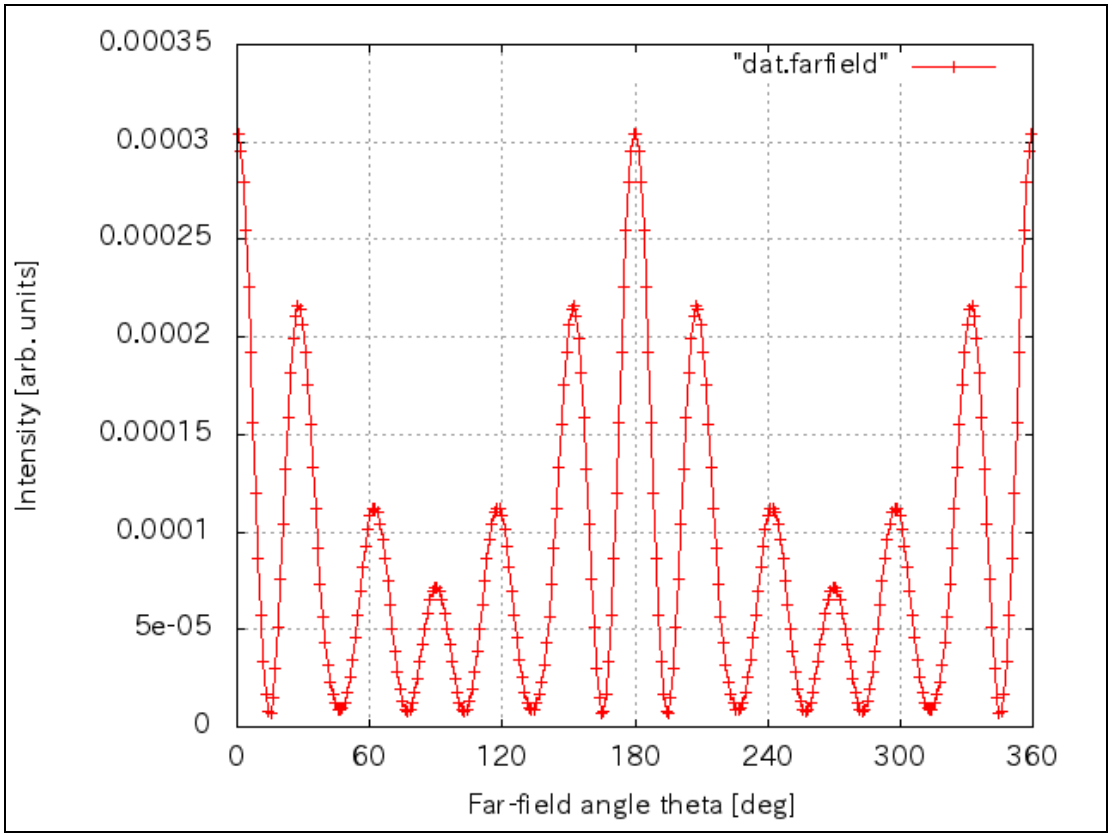



*Fig. 8.13: Far-field intensity distribution of a wave function.*

Go back to table of contents

## 8.5. nearfield.plt

This is a gnuplot script for plotting the near-field data “dat.nearfield”, and is executed as follows:

```
$ nearfield.plt
```

This command creates an image file “dat.nearfield.png” as shown in Fig. 8.14.

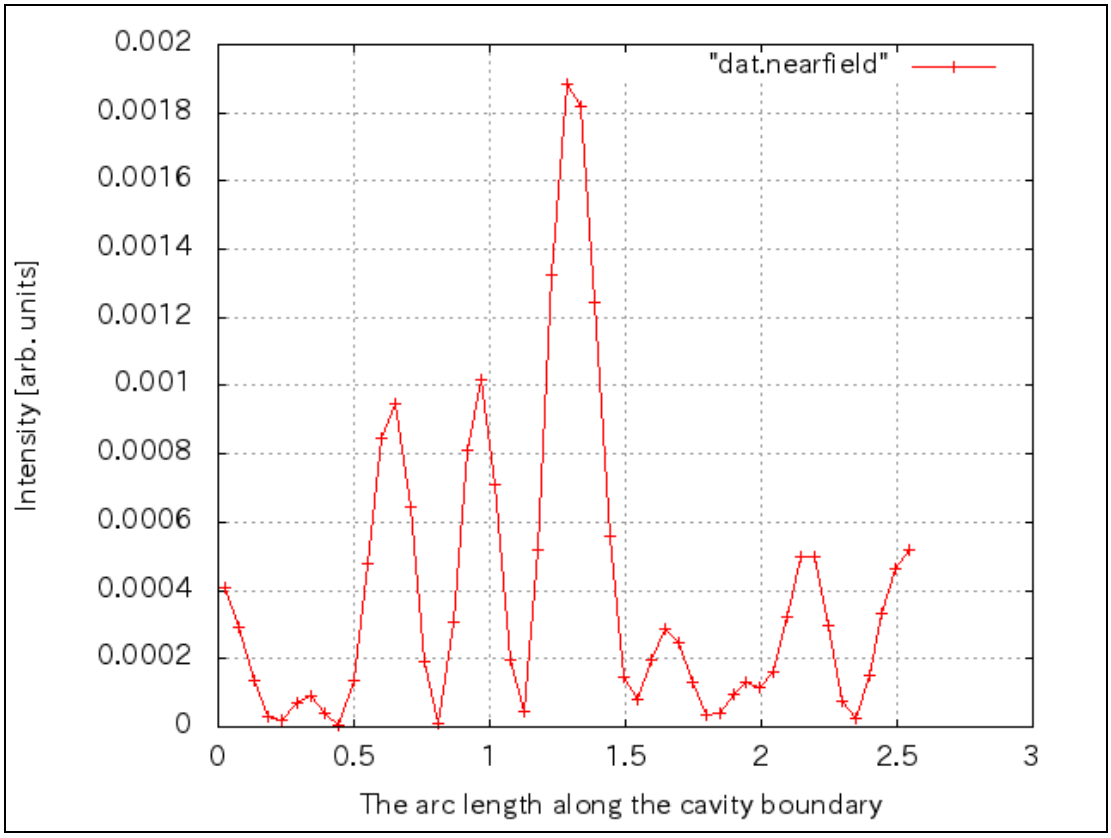



*Fig. 8.14: Intensity distribution along the cavity boundary.*

Go back to table of contents

# 9. Automation tools

**The python scripts, batch_plot.py and autofinder.py, introduced in this section are included only in the Extensive-Tools package.**

## 9.1. batch_plot.py

This python script is used for computing and plotting wave function and Husimi distribution data for each mode listed in the resonance data file.

### 9.1.1 Usage

```
$ batch_plot.py --help
usage: batch_plot.py [-h] --config CONFIG --data DATA [--parallel PARALLEL]
[-v]

Tool to perform parallel-batch-processing of wfunc/husimi.sh for each
resonance in the list.

optional arguments:
-h, --help show this help message and exit
--config CONFIG specify the batch-configuration file
--data DATA specify the resonance-list data file
--parallel PARALLEL specify the number of processes to be executed in
parallel [default: use all cpu cores]
-v, --version show program's version number and exit
```

### 9.1.2. Output files

batch_plot.py creates directories named “mode.[parity info].k=[wave number]”, which contains the following data and image 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 field intensity distribution calculated from the wave function data (in normal scale)

- dat.wfunc_logscale.png : Image file of the field intensity distribution calculated from the wave function data (in log scale)
- dat.farfield.png : Image file of the far-field pattern
- dat.nearfield.png : Image file of the near-field pattern
- dat.husimi.png : Image file of the Husimi distribution

- Shell-scripts:
  - wfunc.sh : Shell-script for generating dat.wfunc, dat.farfield, and dat.nearfield
  - husimi.sh : Shell-script for generating dat.husimi

Go back to table of contents

### 9.1.3. Execution example for the stadium cavity (SYM=2)

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

#### 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 indices inside and outside the cavity
nin=3.3d0
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
```

**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
```

Note that both parity parameters “a” and “b” are provided in data.resonances.ee and batch_plot_config.txt, because SYM=2.

Go back to table of contents

### 9.1.4. Execution example for the D-shaped cavity (SYM=1)

```
batch_plot.py --config batch_plot_config.txt --data data.resonances.e
```

#### batch_plot_config.txt

```
# Batch_plot.py configuration file for data.resonances.e

# Executable for computing wave function data
wfunc_exe=/path/to/wfunc.TM.NBE=100

# Executable for computing Husimi distribution data
husimi_exe=/path/to/husimi.TM.NBE=100

# 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 indices inside and outside the cavity
nin=3.3d0
nout=1.0d0

# Symmetry class parameter
SYM=1

# Parity indices
b=1

# Plot range for the wave function
xmin=-1.2d0
xmax=1.2d0
ymin=-1.2d0
ymax=1.2d0

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

**data.resonances.e**

```
# Resonance data
# polarization=TM
# nin=3.3d0, nout=1.0d0
# nbe=100
# b=1
# ----------------------------------------
# [Re k]  [Im k]  [det]
# ----------------------------------------
5.87145 -0.07435 0.04914715
5.9221 -0.05817 0.048205635
5.95098 -0.06341 0.047387391
6.02331 -0.02653 0.047633906
```

Note that only the parity parameter “b” is provided in data.resonances.e and batch_plot_config.txt, because SYM=1.

Go back to table of contents

## 9.2. autofinder.py

This python script is used for detecting all the resonances in a given wave number range with a given precision.

### 9.2.1. Usage

```
$ autofinder.py --help
usage: autofinder.py [-h] [--workers WORKERS] [--alphax ALPHAX]
[--alphay ALPHAY] [--betax BETAX] [--betay BETAY]
[--epsilonx EPSILONX] [--epsilony EPSILONY]
[--output OUTPUT] [--sorting {0,1,2}] [--ununify]
[--reuse] [--keepall] [--loglevel LOGLEVEL]
[--logdir LOGDIR] [--logname LOGNAME] [--cleanlogs]
executable [inputs [inputs ...]]

Tool for automatically finding minima of a determinant value distribution in
the complex wave number space.

positional arguments:
executable Full-path name to the executable to run
inputs Input parameters of the form 'nin=3.3d0 nout=1.0d0 a=1
b=-1 ...' (default: None)

optional arguments:
-h, --help show this help message and exit
--workers WORKERS, -w WORKERS
Number of concurrent workers (default: 5)
--alphax ALPHAX, -ax ALPHAX
Multiplicative coefficient for determining new dwx
(default: 1.1d0)
--alphay ALPHAY, -ay ALPHAY
Multiplicative coefficient for determining new dwy
(default: 1.1d0)
--betax BETAX, -bx BETAX
Multiplicative coefficient for determining new dx
(default: 0.1d0)
--betay BETAY, -by BETAY
Multiplicative coefficient for determining new dy
(default: 0.1d0)
--epsilonx EPSILONX, -ex EPSILONX
Stop criteria on dx (default: 1.0d-5)
--epsilony EPSILONY, -ey EPSILONY
Stop criteria on dy (default: 1.0d-5)
--output OUTPUT, -o OUTPUT
Base output file name that might be suffixed with
parity characters e/o (suffix format: '.b' for SYM=1,
'.ab' for SYM=2) (default: data.resonances)
--sorting {0,1,2}, -s {0,1,2}
```

```
Field index to used for sorting the resonances
(ascending order): 0=Re(k), 1=Im(k), 2=Det (default:
0)
--ununify Allow dx and dy to be set independently (default:
False)
--reuse Allow to reuse (i.e. read) existing files (default:
False)
--keepall Keep all intermediate files (default: False)
--loglevel LOGLEVEL Set the log level of the log file (default: INFO)
--logdir LOGDIR Set the log directory (default: /tmp/ocms-logs)
--logname LOGNAME Set the string to append to the log file name
(pattern: YYYYMMDD-<logname>.log) (default: ocms)
--cleanlogs Erase all log files in the log directory BEFORE
running the process. Make sure to make a copy of the
log files you want to keep before using this command
(default: False)
```

Go back to table of contents

### 9.2.2. Execution example for the stadium cavity (SYM=2)

```
$ 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
```

**Output file: 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
```

Note that both parity parameters “a” and “b” are provided in the option for autofinder.py and in data.resonances.ee, because SYM=2.

Go back to table of contents

### 9.2.3. Execution example for the D-shaped cavity (SYM=1)

```
$ autofinder.py --keepall /path/to/bin/det.TM.NBE=100 nin=3.3d0 nout=1.0d0 b=1
cx=5.95d0 cy=-0.05d0 dwx=0.1d0 dwy=0.06d0 dx=0.001d0 dy=0.001d0 --epsilonx 1e-
5 --epsilony 1e-5
```

**Output file: data.resonances.e**

```
## ======================================
## Resonance data output by autofinder.py
## ======================================
## Command-line for autofinder.py:
## autofinder.py --keepall /path/to/bin/det.TM.NBE=100 \
## nin=3.3d0 nout=1.0d0 b=1 cx=5.95d0 cy=-0.05d0 \
## dwx=0.1d0 dwy=0.06d0 dx=0.001d0 dy=0.001d0 \
## --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=100
# b=1
# ---------------------------------------
# [Re k]  [Im k]  [det]
# ---------------------------------------
5.87145 -0.07435 0.04914715
5.9221 -0.05817 0.048205635
5.95098 -0.06341 0.047387391
6.02331 -0.02653 0.047633906
```

Note that only the parity parameter “b” is provided in the option for autofinder.py and in data.resonances.e, because SYM=1.

Go back to table of contents

# 10. Data content rules

## 10.1. Data content rules for plot2d.py

### 10.1.1. Overview

plot2d.py can read three types of data format, *complex*, *real* and *mtv*. The complex format is for wave function data (e.g., dat.wfunc), the real format for Husimi distribution data (e.g., dat.husimi), and the mtv format for PlotMTV data (e.g., mtv.wfunc). The mtv format is a legacy option for the data prepared for PlotMTV, which is an obsolete plotting tool.

The data file needs to contain the header and data sections. The lines starting with "#" are recognized as header lines (for the mtv format, "%" is used instead of "#"). In the header section, the parameter values are provided by the following form:

```
# (parameter name) = (parameter value)
```

There are three types of parameters:

- **Mandatory**: the parameters of this type need to be provided for plot2d.py. Otherwise, plot2d.py does not work.
- **Optional**: these parameters have certain effects in plot2d.py, but can be omitted.
- **Commentative**: these parameters have no effects in plot2d.py.

The subsequent two subsections list the possible mandatory and optional parameters.

### 10.1.2. Mandatory parameters for dat.wfunc and dat.husimi

- nx
  - Type: Integer
  - Description: Specifies the number of data points for the x-direction.
- ny
  - Type: Integer
  - Description: Specifies the number of data points for the y-direction.
- xmin
  - Type: Float

    - Description: Specifies the minimum value of the tics for the x-axis of the graph.
- xmax
    - Type: Float
    - Description: Specifies the maximum value of the tics for the x-axis of the graph.
- ymin
    - Type: Float
    - Description: Specifies the minimum value of the tics for the y-axis of the graph.
- ymax
    - Type: Float
    - Description: Specifies the maximum value of the tics for the y-axis of the graph.
- toplable
    - Type: String
    - Description: Specifies the top title text of the graph.
- subtitle
    - Type: String
    - Description: Specifies the sub-title text of the graph.

### 10.1.3. Optional parameters for dat.wfunc and dat.husimi

- vmin
    - Type: Float
    - Description: Specifies the minimum value of the color axis of the graph. Without this parameter, plot2d.py determines the minimum value from the data.
- vmax
    - Type: Float
    - Description: Specifies the maximum value of the color axis of the graph. Without this parameter, plot2d.py determines the maximum value from the data.
- cmin (only for mtv format)

- Type: Float
- Description: Same as vmin.

- cmax (only for mtv format)
  - Type: Float
  - Description: Same as vmax.

- xyratio
  - Type: Float
  - Description: Specifies the ratio “vertical-length/horizontal-length” of the graph box.

- content
  - Type: String (“wavefunction”, “husimi”, or “real”)
  - Description: Specifies the content of the data. The data format is set to be *complex* for “wavefunction”, and *real* for “husimi” and “real”.

Go back to table of contents

### 10.1.4. Example of dat.wfunc

```
$ head -32 dat.wfunc
# content = "wavefunction"
# NBE= 50
# polarization = TM
# nin= 3.2999999999999998
# nout= 1.0000000000000000
# a= 1
# b= 1
# kx= 5.0591400000000002
# ky= -8.7600000000000004E-003
# lambda_in= 0.37634766929363045
# ds_max= 5.2359877559829883E-002
# 0.5*lambda_in/ds_max= 3.5938555133517123
# Dispersion relation = standard
# BZW_HANKEL= 5.0000000000000001E-003
# BZW_MAXINT= 1.0000000000000000E-002
# nx= 500
# ny= 300
# xmin= -2.5000000000000000
# xmax= 2.5000000000000000
# ymin= -1.5000000000000000
# ymax= 1.5000000000000000
# toplabel = "k = 5.0591400000 -0.0087600000 i"
# subtitle = "Parity: a=1, b=1"
# vmax= 2.8461439211148998E-003
# ----------------------------------------
# [Re(phi)] [Im(phi)]
# ----------------------------------------
-0.64235152E-02 -0.29748337E-02
-0.66275478E-02 -0.27163451E-02
-0.68215222E-02 -0.24468454E-02
-0.70048958E-02 -0.21667499E-02
-0.71771441E-02 -0.18765008E-02
```

This file can be opened by plot2d.py by

```
$ plot2d.py dat.wfunc --dataformat complex
```

The “ – –dataformat complex” option can be omitted, because the content is specified as “wavefunction” in the data.

Go back to table of contents

### 10.1.5. Example of dat.husimi

```
$ head -24 dat.husimi
# content = "husimi"
# NBE= 50
# polarization = TM
# nin= 3.2999999999999998
# nout= 1.0000000000000000
# a= 1
# b= 1
# kx= 5.0591400000000002
# ky= -8.7600000000000004E-003
# nx= 200
# ny= 200
# xmin= 0.00000000
# xmax= 1.00000000
# ymin= -1.00000000
# ymax= 1.00000000
# Dispersion relation = standard
# xyratio=1.0
# toplabel = "k = 5.0591400000 -0.0087600000 i"
# subtitle = "Parity: a=1, b=1"
0.4957675637E-02
0.4929944194E-02
0.4902547931E-02
0.4875709246E-02
0.4851191731E-02
```

This file can be opened by plot2d.py by

```
$ plot2d.py dat.husimi --dataformat real
```

The “ – –dataformat real” option can be omitted, because the content is specified as “husimi” in the data.

Go back to table of contents

### 10.1.6. Example of mtv.wfunc

```
$ head -20 mtv.wfunc
$ DATA=CONTOUR
% nx= 500
% ny= 500
% xmin= -3.000000000000000 xmax= 3.000000000000000
% ymin= -2.000000000000000 ymax= 2.000000000000000
% contfill = TRUE
% nsteps=50
% cmin=0
% cmax=2e-7
% toplabel = "nin=3.3, k= 99.981580 -0.005790 i, ee"
% subtitle = "lambda_in=0.0190; ds_max=0.0026; lambda_in/ds_max=7.4074"
0.18955091E-08
0.18024769E-08
0.16331357E-08
0.13977176E-08
0.11150977E-08
0.81191633E-09
0.52052860E-09
0.27584781E-09
0.11125516E-09
```

This file is originally prepared for PlotMTV, and is opened by

```
$ plotmtv mtv.wfunc
```

By plot2d.py, the file can be opened by

```
$ plot2d.py mtv.wfunc --dataformat mtv
```

Go back to table of contents

### 10.1.7. Boundary data for plot2d.py

By the " – –boundary" option, plot2d.py can read the boundary curve data (i.e., data.cavity_boundary) generated by the executable "cavity_data" (see Section 6.5). The following is an example of data.cavity_boundary:

```
$ head -10 data.cavity_boundary
# SYM=2
1      1.9996573249755571      2.6176948307873149E-002      0.99965732497555726
2.6176948307873149E-002 5.2359877559829883E-002 1.0000000000000000
2      1.9969173337331281      7.8459095727844944E-002      0.99691733373312796
7.8459095727844944E-002 5.2359877559829883E-002 1.0000000000000000
3       1.9914448613738105       0.13052619222005157       0.99144486137381038
0.13052619222005157 5.2359877559829883E-002 1.0000000000000000
4       1.9832549075639547       0.18223552549214744       0.98325490756395462
0.18223552549214744 5.2359877559829883E-002 1.0000000000000000
5       1.9723699203976766       0.23344536385590539       0.97236992039767656
0.23344536385590539 5.2359877559829883E-002 1.0000000000000000
6       1.9588197348681931       0.28401534470392259       0.95881973486819305
0.28401534470392259 5.2359877559829883E-002 1.0000000000000000
7       1.9426414910921785       0.33380685923377090       0.94264149109217843
0.33380685923377090 5.2359877559829883E-002 1.0000000000000000
8       1.9238795325112867       0.38268343236508978       0.92387953251128674
0.38268343236508978 5.2359877559829883E-002 1.0000000000000000
9       1.9025852843498607       0.43051109680829508       0.90258528434986063
0.43051109680829508 5.2359877559829883E-002 1.0000000000000000
```

This data file contains the header and data sections. The lines starting with "#" are recognized as header lines. In the header section, **"SYM" is a mandatory parameter**. SYM is the parameter for specifying the symmetry of the cavity (see Section 7.2).

The data section must contain the data consisting of the following 7 columns:

- 1st column: The index n for the boundary element ($1 \leq n \leq NBE$).
- 2nd column: The x-coordinate, x(n), for the center of the n-th boundary element.
- 3rd column: The y-coordinate, y(n), for the center of the n-th boundary element.
- 4th column: The x coordinate, nx(n), of the normal vector at (x(n),y(n)).
- 5th column: The y coordinate, ny(n), of the normal vector at (x(n),y(n)).
- 6th column: The length of the n-th boundary element, ds(n).
- 7th column: The curvature of the boundary curve at (x(n),y(n)).

Go back to table of contents

## 10.2. Data content rules for batch_plot.py

### 10.2.1. Overview

By the "−−config" and "−−data" options, batch_plot.py reads a configuration file (e.g., batch_plot_config.txt) and a resonance data file (e.g., data.resonances), respectively. See Section 9, for the examples of these files.

### 10.2.2. Mandatory parameters for the configuration file for batch_plot.py

- wfunc_exe
  - Type: String
  - Description: Specifies the path of the executable file for computing the wave function (☞).
- husimi_exe
  - Type: String
  - Description: Specifies the path of the executable file for computing the Husimi distribution (☞).
- boundary_data
  - Type: String
  - Description: Specifies the path of the boundary data file (☞).

(☞) The path can be the relative path from the configuration file. When running batch_plot.py, it will be converted to the absolute path automatically.

- wfunc_plot_options
  - Type: String
  - Description: Specifies the options for plotting the wave function (in normal scale) by plot2d.py. For specifying the boundary data, the variable $boundary_data can be used.
  - Example:
    wfunc_plot_options="−−boundary $boundary_data −−linewidth 3"
- wfunc_logplot_options
  - Type: String
  - Description: Specifies the options for plotting the wave function (in log scale) by plot2d.py. For specifying the boundary data, the variable $boundary_data can be used.

- Example:
  wfunc_logplot_options=“ − −boundary $boundary_data − −linewidth 3 − −rightmargin 0.1”

- husimi_plot_options
  - Type: String
  - Description: Specifies the options for plotting the Husimi distribution (in normal scale) by plot2d.py.
  - Example:
    husimi_plot_options=“ − −vscale log”

- nin
  - Type: Float
  - Range: 0<nin<nout<∞
  - Description: Specifies the refractive index inside the cavity.

- nout
  - Type: Float
  - Range: 0<nin<nout<∞
  - Description: Specifies the refractive index outside the cavity.

- SYM
  - Type: Integer
  - Range: 0, 1, 2, or 4
  - Description: Specifies the symmetry of the cavity (see Section 7.2).

- a
  - Type: Integer
  - Range: -1 or 1
  - Description: Specifies the parity with respect to the y axis (see Section 7.6 for the definition of the parity).
    - a=1: Even parity
    - a=-1: Odd parity

- Remark: When SYM=0 or SYM=1, this parameter is absent.

- b
  - Type: Integer
  - Range: -1 or 1
  - Description: Specifies the parity with respect to the x axis for SYM=1 and SYM=2, and to the diagonal x=y for SYM=4 (see Section 7.6 for the definition of the parity).
    - b=1: Even parity
    - b=-1: Odd parity
  - Remark: When SYM=0, this parameter is absent.

- xmin, xmax
  - Type: Float
  - -∞<xmin,xmax<∞
  - Description: xmin and xmax specify the x-range of the area for computing and plotting the wave function data.

- ymin, ymax
  - Type: Float
  - -∞<ymin,ymax<∞
  - Description: ymin and ymax specify the y-range of the area for computing and plotting the wave function data.

- ixmax, iymax

```
- Type: Integer
- 0<ixmin,iymax<&infin;
- Description: Specify the number of data points for computing
and plotting wave function data. The number of data points for
the x direction is ixmax, and that for the y direction is iymax.
```

### 10.2.3. Rules for a resonance data file

The lines starting with “#” are recognized as comments, those starting with other character than “#” are recognized as data. The data must consist of three columns:

- 1st column: The real part of the wave number (Re k)
- 2nd column: The imaginary part of the wave number (Im k)
- 3rd column: The determinant value (det)

Go back to table of contents

# 11. Definition of the cavity shape

## 11.1. Overview

The cavity shape is defined by the code named def_cavity_SYM{0,1,2,4}_[cavity shape].F in $OCMS/src/def_cavities. This code defines the cavity boundary points in the x-y plane, the boundary elements along the boundary, normal vectors and curvatures at the boundary. Also, the parameters related to the cavity shape are specified in this code.

The OCMS package includes the definition codes for the following cavities:

- Asymmetrically cut disk cavity
- Asymmetric limaçon cavity
- Rounded triangular cavity
- D-shaped cavity
- Annular cavity
- Cardioid cavity
- Deformed circular cavity with $D_2$ symmetry
- Elliptic cavity
- Flattened quadrupole cavity
- Stadium cavity
- Sinai's cavity (with rounded corners)

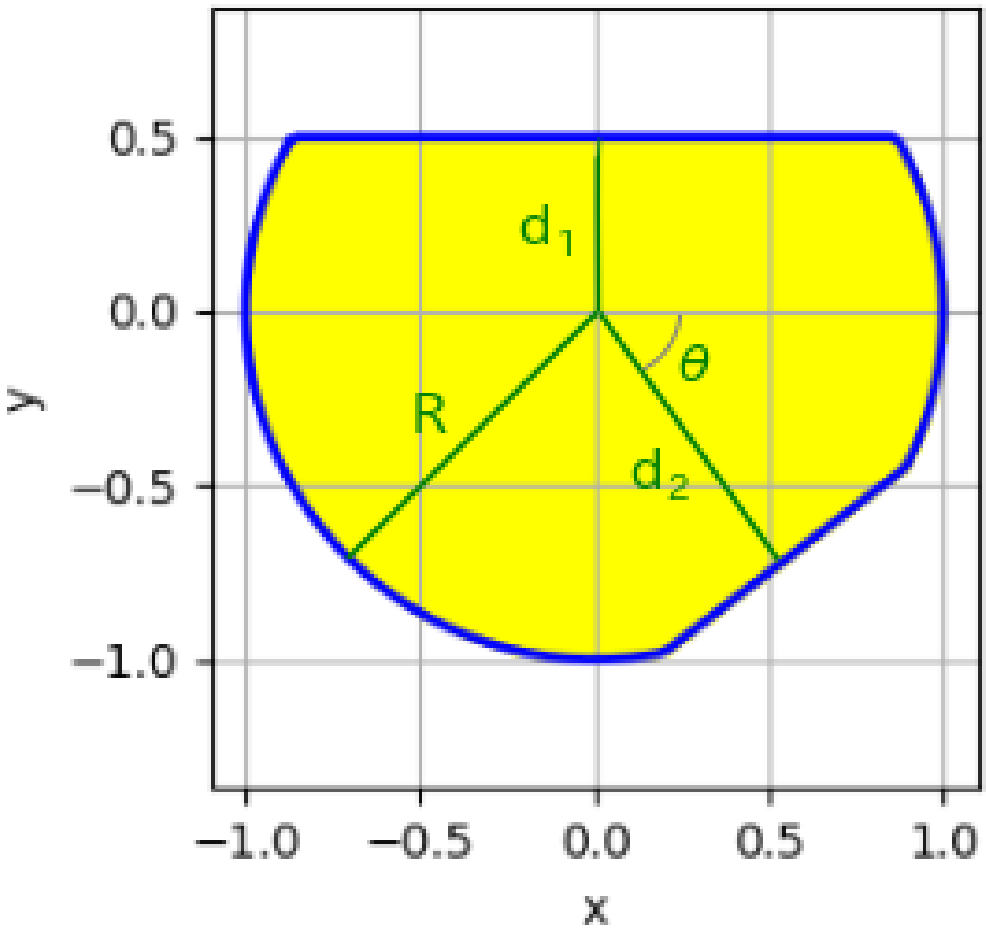
The detailed information of these cavities are given in the following pages.

Go back to table of contents

## 11.2. Detailed information on the cavity shapes

### Asymmetrically cut disk cavity

- Definition code: def_cavity_SYM0_asym_cut_disk.F
- Parameters: $0<R<\infty$, $0\leq d1\leq R$, $0\leq d2\leq R$, $0\leq\theta\leq\pi$.
- Default: R=1, d1=0.5, d2=0.9, $\theta=\pi(7-3\sqrt{5})$.




- References:
  - L.A. Bunimovich, “On the ergodic properties of nowhere dispersing billiards”, Commun. Math. Phys. **65**, 295-312 (1979).

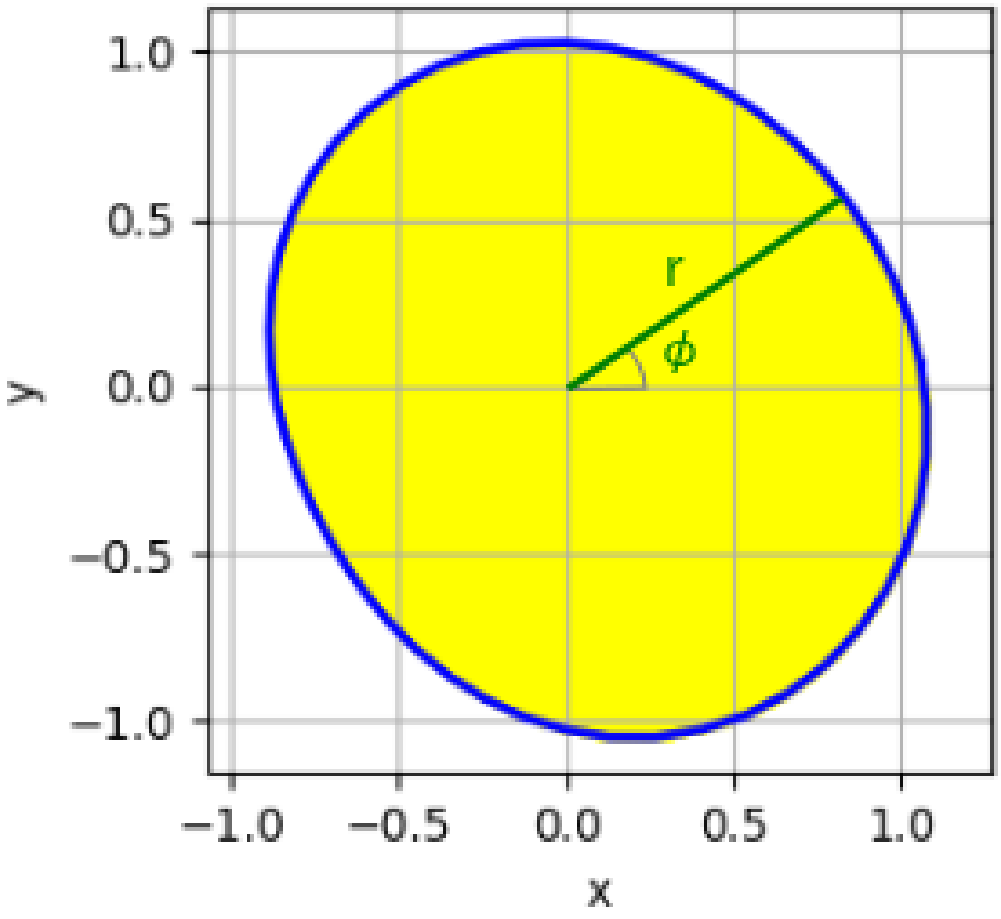
Go back to table of contents

### Asymmetric limaçon cavity

- Definition code: def_cavity_SYM0_asym_limacon.F
- Boundary curve:

$$r = r(\varphi) = R\,[\,1 + \varepsilon_1 \cos(\varphi) + \varepsilon_2 \cos(2\varphi+\delta)\,],$$
$$\text{with } \delta=\pi(\sqrt{5}-1)/2.$$

- Parameters: $0<R<\infty$, $0\leq\varepsilon_1<\infty$, $0\leq\varepsilon_2<\infty$.
- Default: $R=1$, $\varepsilon_1=0.1$, $\varepsilon_2=0.075$.

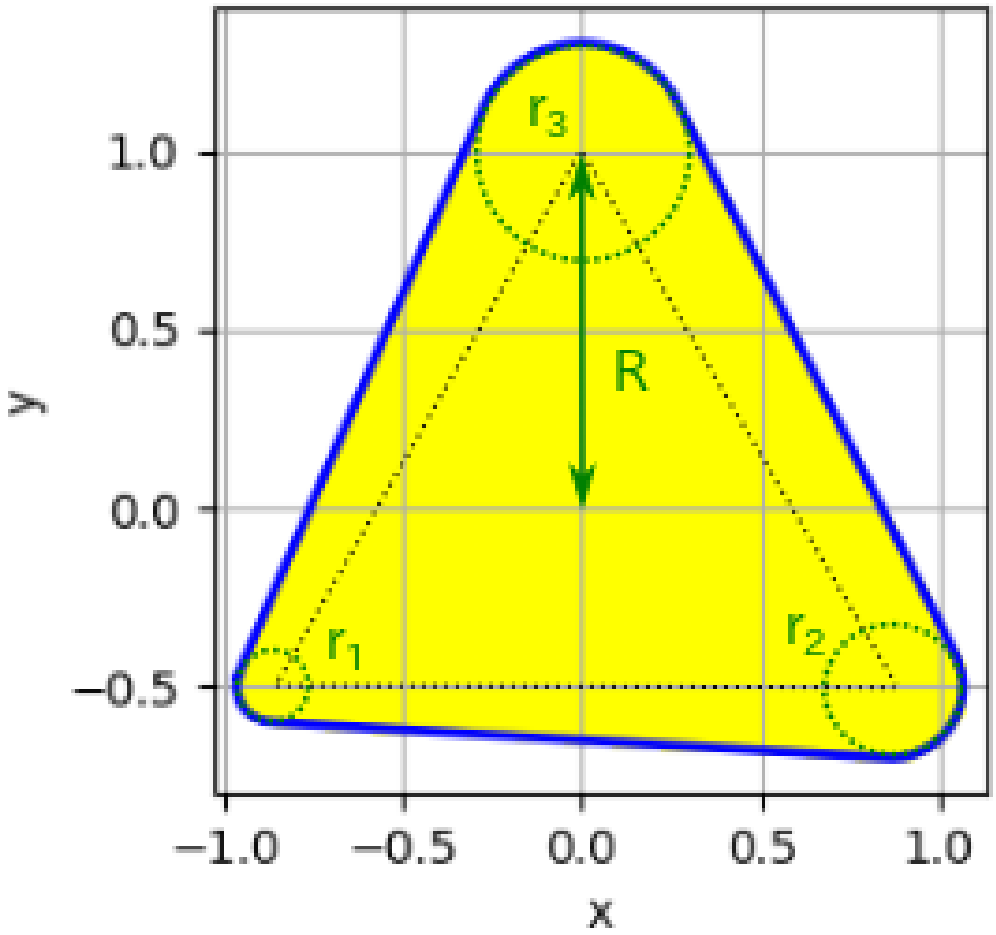



- References:
  - J. Wiersig, A. Eberspächer, J.-B. Shim, J.-W. Ryu, S. Shinohara, M. Hentschel, and H. Schomerus, “Nonorthgonal pairs of copropagating optical modes in deformed microdisk cavities”, Phys. Rev. A **84**, 023845 (2011).

Go back to table of contents

### Rounded triangular cavity

- Definition code: def_cavity_SYM0_rounded_triangle.F
- Parameters: $0<R<\infty$, $0\leq r_1$, $0\leq r_2$, $0\leq r_3$.
- Default: R=1, $r_1$=0.1, $r_2$=0.2, $r_3$=0.3.

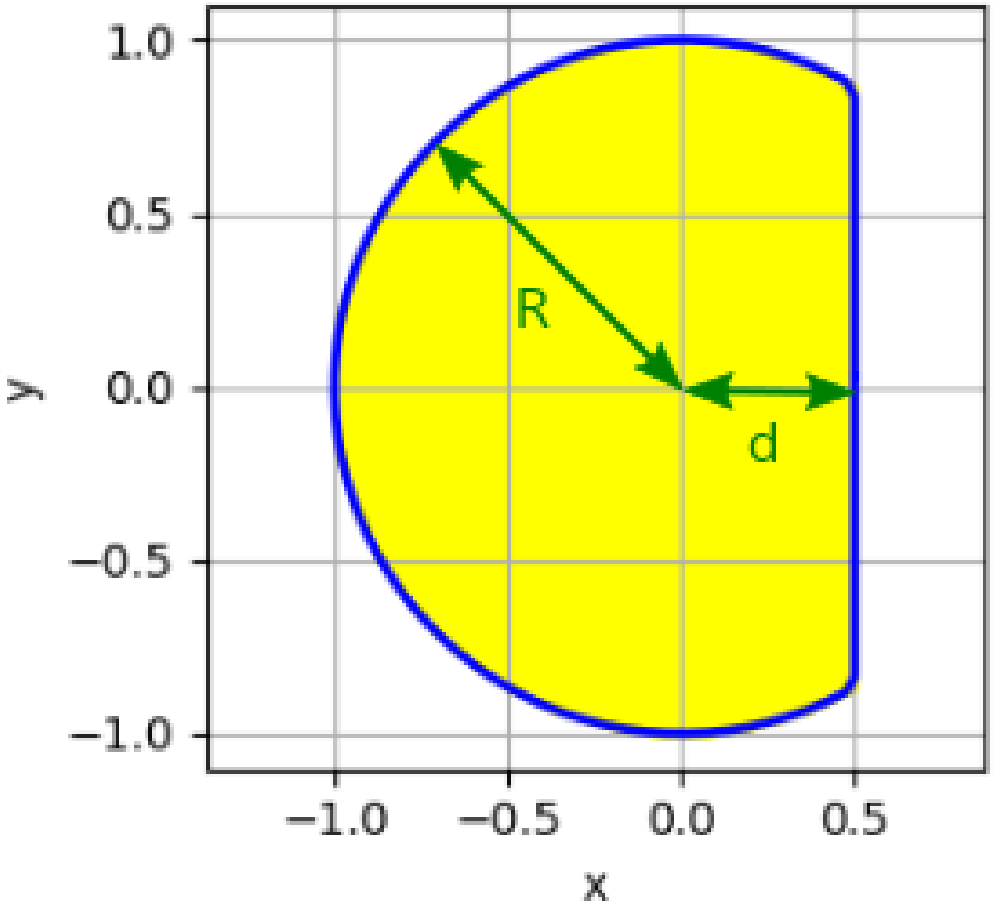



- References:
  - L.A. Bunimovich, “On the ergodic properties of nowhere dispersing billiards”, Commun. Math. Phys. **65**, 295-312 (1979).

Go back to table of contents

### D-shaped cavity

- Definition code: def_cavity_SYM1_D-shape.F
- Parameters: $0 \leq R < \infty$, $0 \leq d \leq R$.
- Default: R=1, d=0.5.

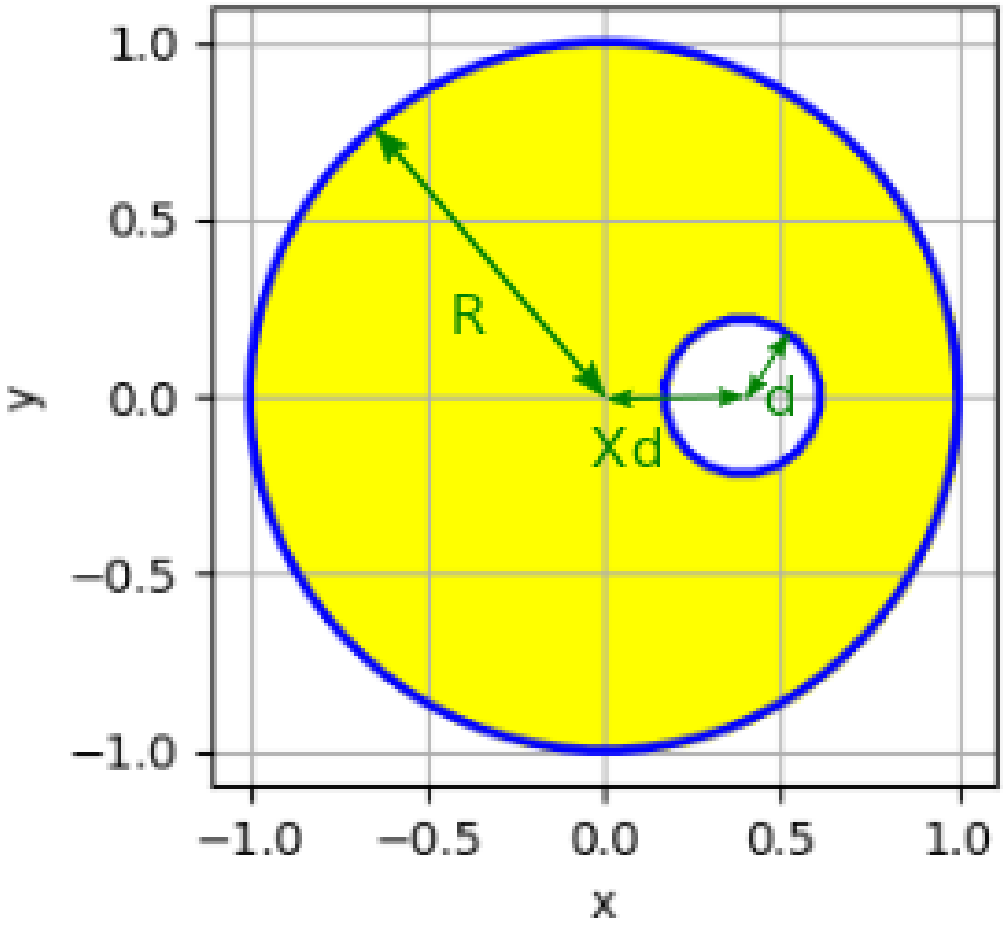



- References:
  - L.A. Bunimovich, “On the ergodic properties of nowhere dispersing billiards”, Commun. Math. Phys. **65**, 295-312 (1979).
  - B. Redding, A. Cerjan, X. Huang, M.L. Lee, A.D. Stone, M.A. Choma, and H. Cao, “Low spatial coherence electrically pumped semiconductor laser for speckle-free full-field imaging”, PNAS **112**, 1304-1309 (2015).

Go back to table of contents

### Annular cavity

- Definition code: def_cavity_SYM1_annular.F
- Parameters: $0 \leq R < \infty$ $0 \leq d$, $0 \leq Xd < R-d$.
- Default: R=1, d=0.22, Xd=0.391.




- References:
  - N. Saito, H. Hirooka, J. Ford, F. Vivaldi, and G.H. Walker, “Numerical study of billiard motion in an annulus bounded by non-concentric circles”, Physica D **5**, 273-286 (1982).
  - M. Hentschel, H. Schomerus, and R. Schubert, “Husimi functions at dielectric interfaces: Inside-outside duality for optical systems and beyond”, Europhys. Lett. **62**, 636-642 (2003).
  - A. Bäcker, R. Ketzmerick, S. Löck, J. Wiersig, M. Hentschel, “Quality factors and dynamical tunneling in annular microcavities”, Phys. Rev. A **79**, 063804 (2009).

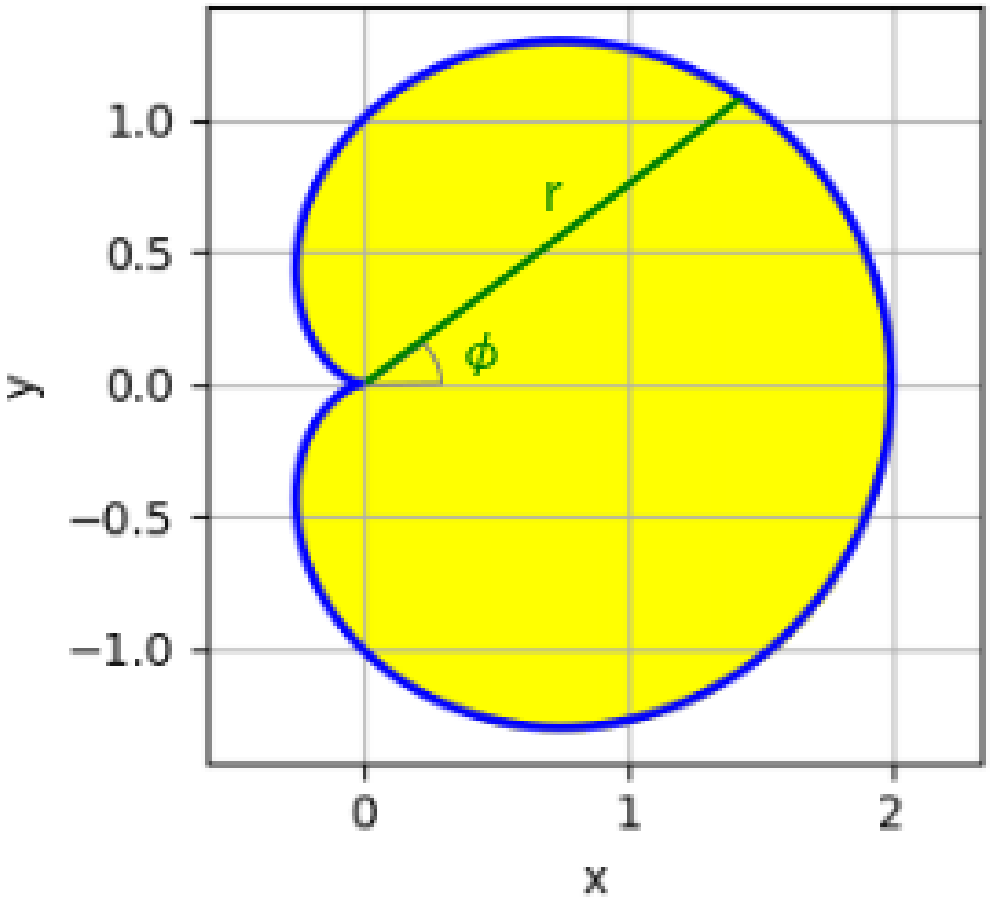
Go back to table of contents

### Cardioid cavity

- Definition code: def_cavity_SYM1_cardioid.F
- Boundary curve:

$$r=r(\varphi) =R\ [\ 1+\varepsilon\cos(\varphi)\ ]$$

- Parameters: $0 \leq R < \infty$.
- Default: R=1.




- References:
  - M. Wojtkowski, “Principles for the design of billiards with nonvanishing Lyapunov exponents”, Commun. Math. Phys. **105**, 391-414 (1986).

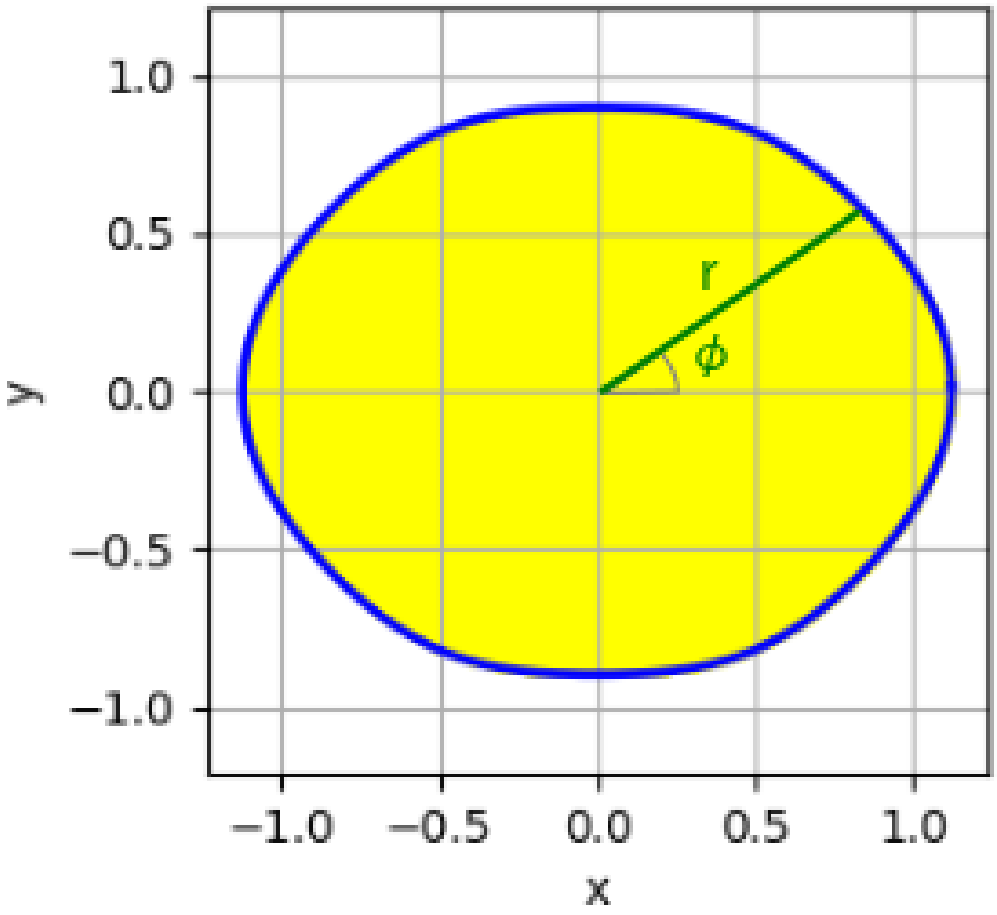
Go back to table of contents

### Deformed circular cavity with $D_2$ symmetry

- Definition code: def_cavity_SYM2_D2_deformed_circle.F
- Boundary curve:

$$r = r(\varphi) = R\,[\,1 + \varepsilon_1\cos(2\varphi) + \varepsilon_2\cos(4\varphi) + \varepsilon_3\cos(6\varphi)]$$

- Parameters: $0<R<\infty$, $0\leq\varepsilon_1$, $0\leq\varepsilon_2$, $0\leq\varepsilon_3$.
- Default: $R=1$, $\varepsilon_1=0.1$, $\varepsilon_2=0.01$, $\varepsilon_3=0.012$.
- Note: the cavity with $\varepsilon_2=\varepsilon_3=0$ reduces to the quadrupole-deformed cavity.




- References:
  - S. Shinohara, T. Harayama, T. Fukushima, M. Hentschel, T. Sasaki, and E.E. Narimanov, “Chaos-assisted directional light emission from microcavity lasers”, Phys. Rev. Lett. **104**, 163902 (2010).

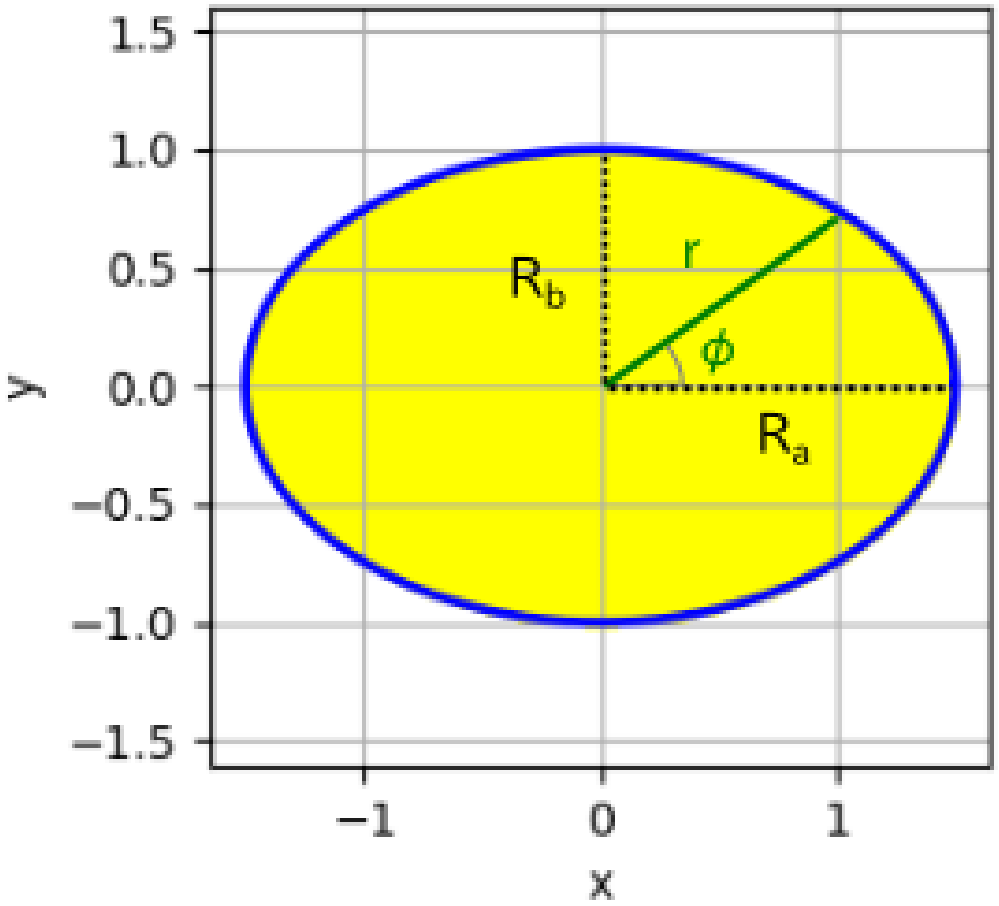
Go back to table of contents

### Elliptic cavity

- Definition code: def_cavity_SYM2_ellipse.F
- Boundary curve:

$$r=r(\varphi) = R_a R_b / [ R_b^2 \cos^2\varphi + R_a^2 \sin^2\varphi ]^{1/2}$$

- Parameters: $0 \leq R_a < \infty$, $0 \leq R_b < \infty$.
- Default: $R_a$=1.5, $R_b$=1.0.



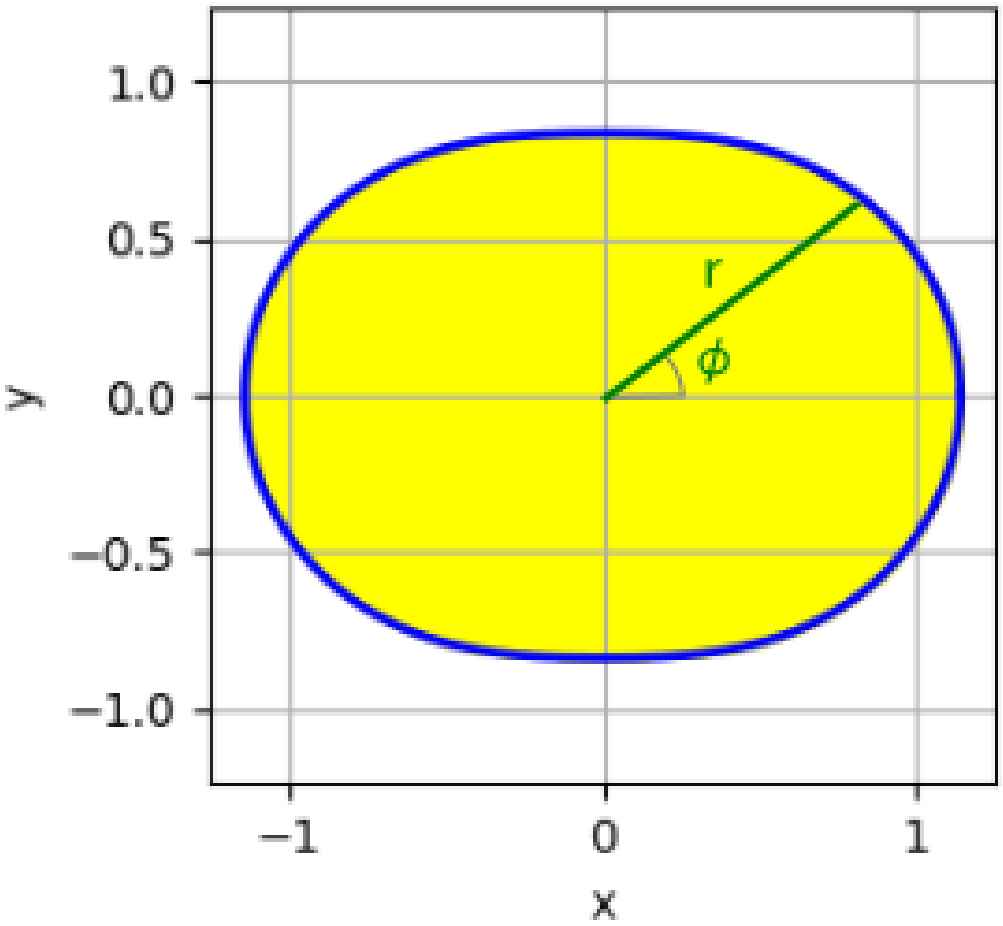

Go back to table of contents

### Flattened quadrupole cavity

- Definition code: def_cavity_SYM2_D2_flattened_quadrupole.F
- Boundary curve:

$$r = r(\varphi) = R\sqrt{1 + 2\varepsilon\cos(2\varphi)}$$

- Parameters: $0<R<\infty$, $0\leq\varepsilon<\infty$.
- Default: R=1, ε=0.15.

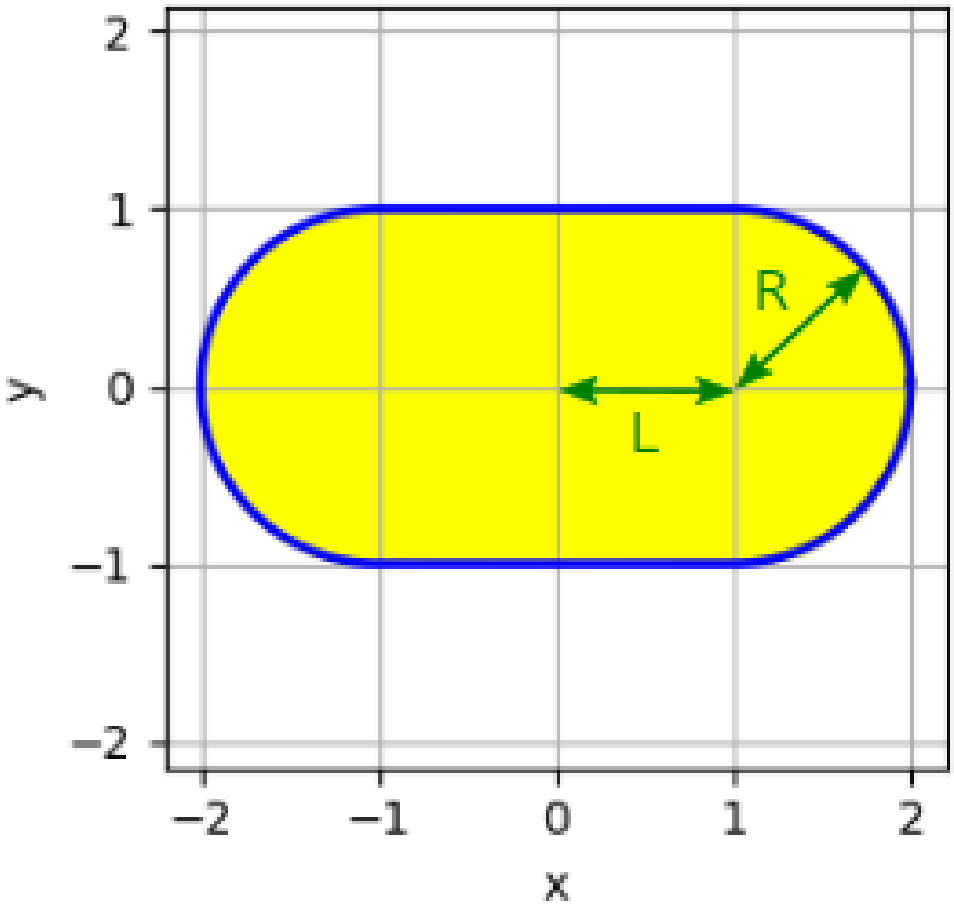



- References:
  - C. Gmachl, F. Capasso, E.E. Narimanov, J.U. Nöckel, A.D. Stone, J. Faist, D.L. Sivco, A.Y. Cho, “High-power directional emission from microlasers with chaotic resonators”, Science **280**, 1556-1564 (1998).

Go back to table of contents

### Stadium cavity

- Definition code: def_cavity_SYM2_stadium.F
- Parameters: $0 \leq R < \infty$, $0 \leq L < \infty$
- Default: R=L=1.




- References:
  - L.A. Bunimovich, “On the ergodic properties of nowhere dispersing billiards”, Commun. Math. Phys. **65**, 295-312 (1979).
  - T. Harayama, P. Davis, and K.S. Ikeda, “Stable oscillations of a spatially chaotic wave function in a microstadium laser”, Phys. Rev. Lett. **90**, 063901 (2003).
  - T. Fukushima and T. Harayama, “Stadium and quasi-stadium laser diodes”, IEEE JSTQE **10**, 1039-1051 (2004).
  - M. Lebental, J.S. Lauret, R. Hierle, and J. Zyss, “Highly directional stadium-shaped polymer microlasers”, Appl. Phys. Lett. **88**, 031108 (2006).

Go back to table of contents

### Sinai's cavity (with rounded corners)

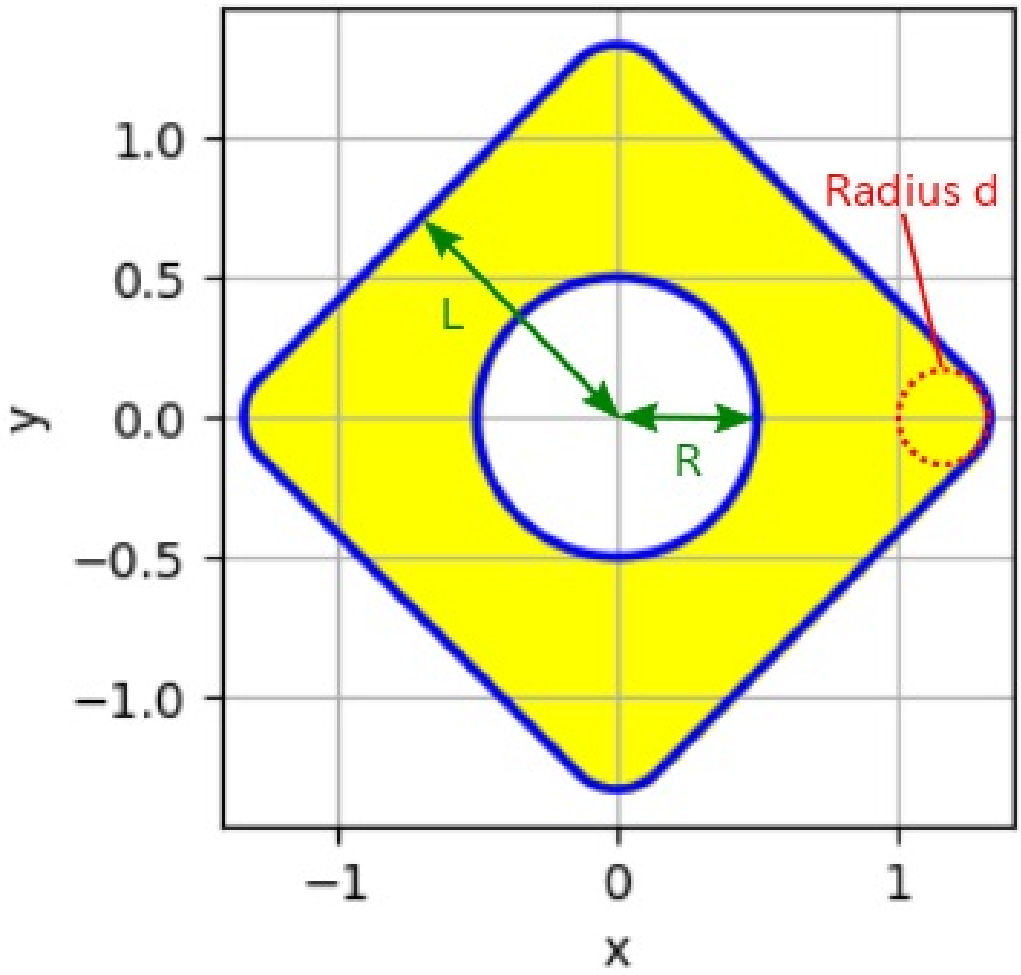
- Definition codes:
  i. def_cavity_SYM4_Sinai.F: Version that takes into account $C_{4v}$ symmetry of Sinai's cavity (i.e., mirror symmetry with respect to the diagonal $x = y$ and the $y$-axis).

  ii. def_cavity_SYM2_Sinai.F: Version that takes into account $C_{2v}$ symmetry of Sinai's cavity (i.e., mirror symmetry with respect to $x$- and $y$-axes).

  iii. def_cavity_SYM0_Sinai.F: Version that does not take into account symmetry of Sinai's cavity.

- Parameters: $0 \leq (R+2d) < \sqrt{(2)}L < \infty$

- Default: L=1, R=0.5, d=0.2.




- References:
  - Ya.G. Sinai, "On the foundations of the ergodic hypothesis for a dynamical system of statistical mechanics", Sov. Math. Dokl. **4**, 1818 (1963).

- Symmetry classes of Sinai's cavity

Sinai's cavity has $C_{4v}$ point group symmetry. This implies the resonant modes to be classified into six symmetry classes, corresponding to irreducible representations of the point group $C_{4v}$. The six symmetry classes are shown in Fig. 11.1.

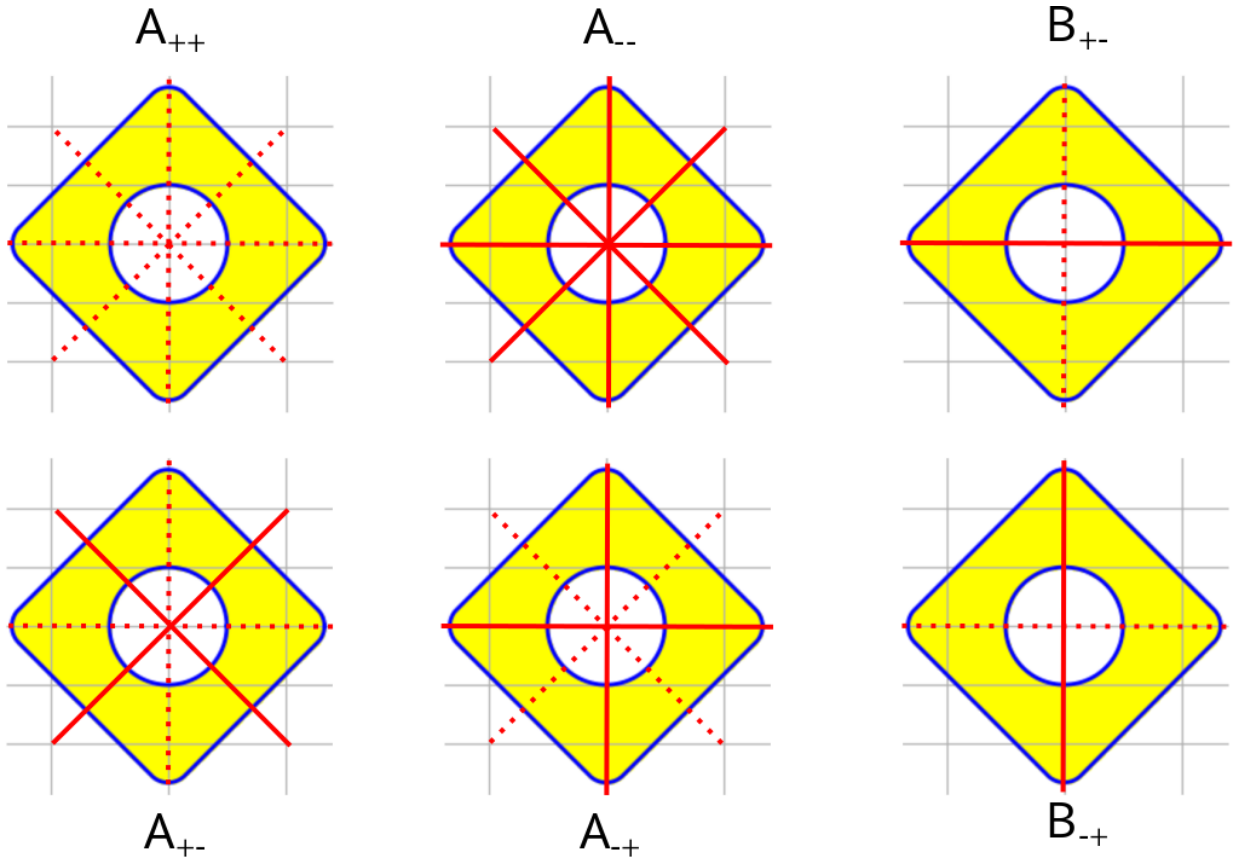



*Fig. 11.1: Six symmetry classes for Sinai's cavity.*

The solid line represents even symmetry, whereas the dotted line represents odd symmetry.

The modes in the symmetry classes $B_{+-}$ and $B_{-+}$ are degenerate. Namely, if you find a $B_{+-}$ mode with a wave number k, there is a $B_{-+}$ mode with the exactly same wave number k, and vice versa. This degeneracy is due to the fact that the eigen function of a $B_{+-}$ mode can be reproduced by a superposition of rotated eigen functions of a $B_{-+}$ mode.

For the detection of *A* modes, the cavity shape definition code with SYM=4 (i.e., def_cavity_SYM4_Sinai.F) is suitable. Meanwhile, for the detection of *B* modes, the cavity shape definition code with SYM=2 (i.e., def_cavity_SYM2_Sinai.F) is suitable. The code with SYM=0 (i.e., def_cavity_SYM0_Sinai.F) can be also used for simultaneous detection of *A* and *B* modes. However, it fails to resolve the degeneracy of $B_{+,-}$ and $B_{-,+}$ modes mentioned above.

Go back to table of contents

## 11.3. Adding a new cavity shape

### 11.3.1. Overview

The user can define an arbitrary new cavity shape by writing a cavity shaped definition code (i.e., def_cavity_SYM{0,1,2,4}_[new cavity shape].F). In this code, the following four subroutines must be defined:

- set_parameters()
- test_cavity_symmetry(symmetry)
- def_bndry(nbe,ds,kappa,xl,yl,nx,ny)
- indicator_func(x,y,flag)

The program main.cavity.F is used for checking whether def_bndry() and indicator_func() are defined as you wished. This program can be compiled as follows:

```
cd $OCMS
make cavity
```

Then, you get the executable file “cavity_data” in the directory $OCMS. By running this file, you get “data.cavity_boundary” and “data.cavity_domain”. The former is the output data of def_bndry() and the latter is the output data of indicator_wfunc(). These data can be visualized by boundary_plot.py (see the instructions of boundary_plot.py for how to use it).

In the following, the roles of the four subroutines are described:

### 11.3.2. set_parameters()

Parameter values for the cavity shape are specified in this subroutine. For example, in case of the D-shaped cavity, the parameters are the radius of the circular part, R, and the distance of the flat part from the origin, d. This subroutine is called only in def_bndry() and indicator_func().

### 11.3.3. test_cavity_symmetry()

In this subroutine, the symmetry class of the cavity (CAVITY_SYM_CLASS) is defined:

- CAVITY_SYM_CLASS=0: The cavity has no mirror symmetry.
- CAVITY_SYM_CLASS=1: The cavity is symmetric with respect to the x-axis.
- CAVITY_SYM_CLASS=2: The cavity is symmetric with respect to both x- and y-axis.
- CAVITY_SYM_CLASS=4: The cavity is symmetric with respect to the diagonal x=y and the y-axis.

This subroutine checks the consistency between the value of SYM and the cavity shape, both defined in build-params.mk.

### 11.3.4. def_bndry()

The subroutine def_bndry(nbe,ds,kappa,xl,yl,nx,ny) defines quantities pertinent to the cavity shape. For CAVITY_SYM_CLASS=4, the cavity shape is defined for the first quadrant region below the diagonal x=y (i.e., x,y≥0 and y≤x). For CAVITY_SYM_CLASS=2, the cavity shape is defined for the first quadrant (i.e., x,y≥0). For CAVITY_SYM_CLASS=1, the cavity shape is defined for the upper half plane (i.e., y≥0). For CAVITY_SYM_CLASS=0, the cavity shape is defined for the full x-y plane.

nbe is the number of boundary elements, whose value is input when this subroutine is called.

The variables ds(1:nbe), kappa(1:nbe), xl(1:nbe), yl(1:nbe), nx(1:nbe), ny(1:nbe) are defined in this subroutine. The meaning of these variables are as follows:

- ds(n): The length of the n-th boundary element.
- kappa(n): The curvature at the center of the n-th boundary element.
- (xl(n),yl(n)): The x and y coordinates of the center of the n-th boundary element.
- (nx(n),ny(n)): The x and y components of the normal vector (pointing outside the cavity) at the center of the n-th boundary element.

This subroutine is called in most of the main programs (main.cavity.F, main.det.F, main.wfunc.F, main.husimi.F).

### 11.3.5. indicator_func()

This subroutine checks whether a given point (x,y) is inside or outside the cavity. If inside (resp. outside), it returns flag=1 (resp. flag=0). This subroutine is called only in main.wfunc.F.

Go back to table of contents

# Appendix 1: Install gfortran

The following is an installation procedure for Ubuntu 18.04 LTS:

```
$ sudo apt update
$ sudo apt install gfortran
```

Go back to table of contents

# Appendix 2: Install LAPACK and BLAS

The following is an installation procedure for Ubuntu 18.04 LTS:

## 1. Installation of cmake

For building and installing LAPACK and BLAS, cmake is required.

```
sudo apt install cmake
```

## 2. Download LAPACK (containing BLAS)

Download the LAPACK source codes “lapack-3.8.0.tar.gz” from: http://www.netlib.org/lapack/

## 3. Extract files

Move the downloaded file to the working directory, and extract files:

```
$ tar xzvf lapack-3.8.0.tar.gz
```

This command creates a new directory “lapack-3.8.0”.

## 4. Build and install

```
$ cd lapack-3.8.0
$ mkdir build
$ cd build
$ cmake -DCBLAS=ON -DLAPACKE=ON -DLAPACKE_WITH_TMG=ON ..
$ cmake --build .
```

Note that the build options for cmake have the following effects:

- -DCBLAS=ON: build CBLAS
- -DLAPACKE=ON: build LAPACKE
- -DLAPACKE_WITH_TMG=ON: build LAPACKE with TMG (needed to get tmglib)

Install the built components in /usr/local (default location):

```
$ sudo cmake --build . --target install
```

Note that several subfolders will be automatically generated such as /usr/local/lib, /usr/local/include, if they do not already exist.

Go back to table of contents

# Appendix 3: Install SLATEC

The following is an installation procedure for Ubuntu 18.04 LTS:

## 1. Download SLATEC

Download the SLATEC source codes “slatec_src.tgz” and “slatec4linux.tgz” from:
http://www.netlib.org/slatec/

## 2. Extract files

Move the downloaded files to a working directory, and extract files:

```
$ cd <working_directory>
$ tar xzvf slatec_src.tgz
```

By the above command, a new directory “src/” is created.

```
$ cp slatec4linux.tgz ./src/
$ cd src/
$ tar xzvf slatec4linux.tgz
```

## 3. Build and install

Build the source codes:

```
$ FC=gfortran make
```

For the installation, you need to first insure that man1 folder exists in /usr/local/man/.

```
$ sudo mkdir -p /usr/local/man/man1
$ sudo make install
```

Go back to table of contents

# Appendix 4: Install Python3 environment via anyenv

## 1. Installation of curl

curl is used for downloading an installation script.

```
$ sudo apt install curl
```

## 2. Installation of pyenv

Type in a terminal:

```
$ curl https://pyenv.run | bash
```

Edit ~/.bashrc and add these lines at the end if not automatically added:

```
export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

Reload .bashrc:

```
$ source .bashrc
```

Now you can use pyenv:

```
$ pyenv update
$ pyenv install -l
```

- Reference:
  - https://github.com/pyenv/pyenv-installer

## 3. Installation of python environment using miniconda

Install the miniconda3-4.3.30:

```
$ pyenv install miniconda3-4.3.30
```

Check the installed python

```
$ pyenv versions
* system
miniconda3-4.3.30
```

Default global and local python version is “system”, where “system” is system default (standalone) python.

```
$ cd <working directory>
$ pyenv global
system

$ pyenv local
system

$ python
Python 2.7.16 (default, Mar 4 2019, 09:02:22)
[GCC 4.2.1 Compatible Apple LLVM 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you want to use another python environment in your working directory, set as follows:

```
$ cd <working directory>
$ pyenv local miniconda3-4.3.30
```

Check the local python version:

```
$ pyenv global
system

$ pyenv local
miniconda3-4.3.30

$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

## 4. Install numpy and matplotlib

You can install numpy and matplotlib using “conda” command in miniconda environment.

Update conda. Proceed with “y” (yes), if you are asked whether to update some packages.

```
$ conda update conda
```

Update python packages in miniconda. Proceed with “y” (yes), if you are asked whether to update some packages.

```
$ conda update --all
```

Install numpy and matplotlib:

```
$ conda install numpy matplotlib
```

# Appendix 5: Install gnuplot

The following is an install procedure for Ubuntu 18.04 LTS:

```
$ sudo apt update
$ sudo apt install gnuplot-x11
```

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