gtrace Documentation

Release 0.1.0

Yoichi Aso

November 08, 2012

CONTENTS

1 Introduction **3**

2 Basic Concepts **5**
- 2.1 2D plane . . . 5
- 2.2 Direction . . . 5
- 2.3 Beam . . . 6

3 API References **7**
- 3.1 gtrace Package . . . 7

4 Indices and tables **21**

Python Module Index **23**

Index **25**

Contents:

CHAPTER

ONE

INTRODUCTION

gtrace is a python package to trace the propagation of Gaussian beams among optical components such as mirrors and lenses. The features of gtrace include:

- Automatically track the Gaussian beam propagation, i.e. q-parameter change, using the ABCD matrix method.
- Reflection and refraction at interface surfaces are properly treated.
- Automatically track the optical distance traveled by a beam through dielectric media.
- Sequential or non-sequential trace modes are available.
- Exporting the results to DXF files

The main motivation behind the development of gtrace was to help the design of the optical layout for the KAGRA interferometer. The task is not so trivial because we had to satisfy many constraints at the same time. Manually placing mirrors on a CAD and adjusting the distances between the mirrors and their orientations was not an option. We needed a way to automatically adjust the optical layout with a computer to satisfy our requirements. In order to do this, we needed a tool to represent an optical layout convenient for computer processings. gtrace represents mirrors and beams as class objects in python. Then the propagation of the beams can be treated as interactions between the beams and the mirrors. This way, we can automate the optimization of the KAGRA optical layout.

The main ingredients of gtrace are mirrors and beams. A mirror is represented as an instance of `Mirror` class. You place mirrors in a 2D plane. You can set various properties of a mirror such as size, curvature, reflectivities, wedge angle etc. Then you launch a beam, an instance of `GaussianBeam` class, from a certain point in the 2D plane. As the beam hits mirrors, it is divided into several sub-beams, such as reflections and deflected beams. These newly generated beam objects are available to you for further propagation. In the non-sequential mode, these sub-beams are automatically propagated until one of the termination conditions is satisfied. The termination conditions include, not hitting any mirror, power is below a certain threshold and so on. At the end of the propagation, you will have a collection of beam objects generated by the collision with the mirrors. You can export the mirrors and the beams into a DXF file.

gtrace is at this moment limited to 2D optical layouts. This limitation might be lifted in the future.

CHAPTER

TWO

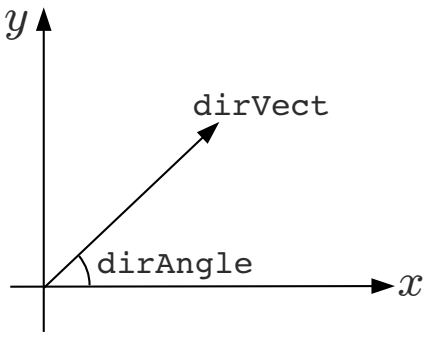
BASIC CONCEPTS

In this section, basic concepts of gtrace will be introduced.

2.1 2D plane

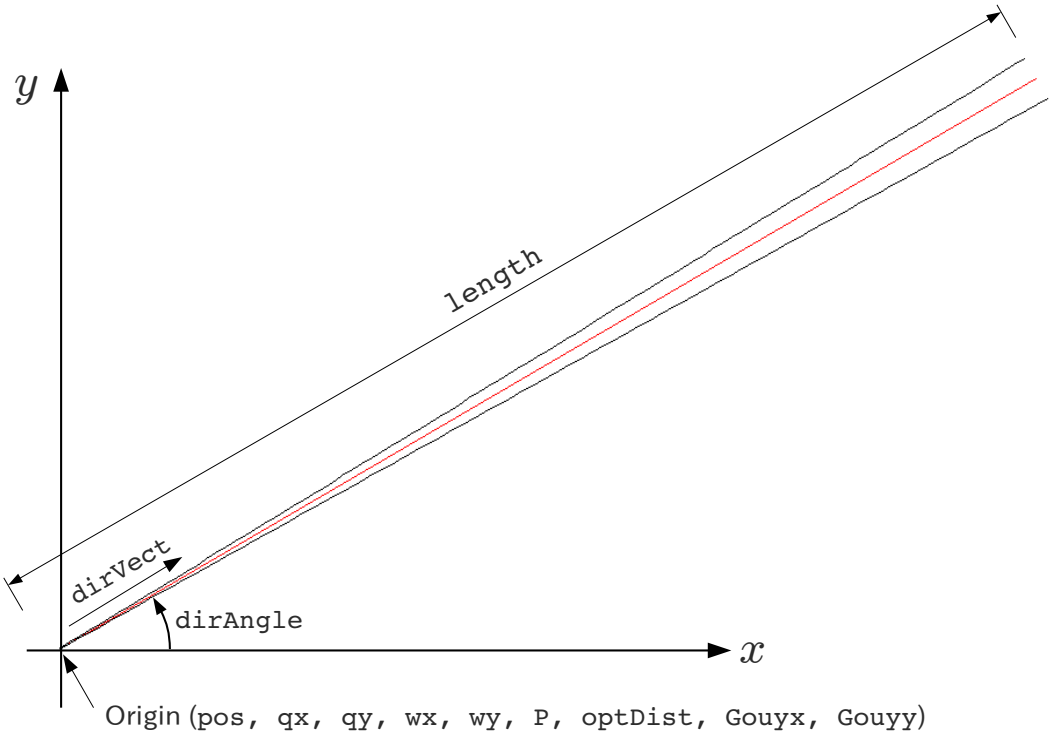
In gtrace world, an optical system will be placed on a two dimentional plane. A location on the plane is specified by a set of Cartesian coordinates (x, y). This just a normal x-y plane. The origin of the axes is at the lower left of the plane. The X-axis extends horizontally to the right. The Y-axis goes up vertically. Nothing more to add here.

2.2 Direction



While working with optical layouts, one often has to specify a direction in the 2D plane such as the orientation of a mirror or the propagation direction of a beam. In gtrace, in most cases, a direction can be specified in two ways. One way is to use an angle measured from the X-axis in counter clockwise (`dirAngle` in the figure above). The other way is to use a 2D vector of length 1. If a direction can be specified either way, you only have to specify it in one of those methods. For example, the `GaussianBeam` class has an attribute called `dirVect`. It holds a 2D vector in the form of `numpy.Array`. The `GaussianBeam` class also has an attribute called `dirAngle`, which holds the angle of the beam propagation direction measured from the X-axis in radian. When one of the two attributes is changed, the other is updated automatically to be consistent with the modification. Therefore, you don't have to worry about the consistency. For the direction vector, it is also automatically normalized. Therefore, you can assign it a vector of any norm.

2.3 Beam



A Gaussian beam is represented by an instance of `GaussianBeam` class. The most fundamental properties of a beam is its position (`pos`) and the direction of propagation (`dirVect` or `dirAngle`).

CHAPTER

THREE

API REFERENCES

3.1 gtrace Package

3.1.1 gtrace Package

This package provides necessary classes and functions for tracing the propagation of Gaussian beams among mirrors and lenses.

3.1.2 beam Module

gtrace.beam

A module to define GaussianBeam class.

class gtrace.beam.**GaussianBeam**(*q0=2.952624674426497j, q0x=False, q0y=False, pos=[0.0, 0.0], length=1.0, dirAngle=0.0, dirVect=None, wl=1.064e-06, P=1.0, n=1.0, name='Beam', layer='main_beam'*)

Bases: traits.has_traits.HasTraits

This is a class to represent a Gaussian beam. A GaussianBeam object has its origin (pos) and a propagation direction (dirVect or dirAngle). A GaussianBeam is characterized by q-parameter(s) at its origin. The beam can be either circular or elliptic. In order to deal with elliptic beams, some parameters are stored in pairs like (q0x, q0y). x and y denote the axes of the cross section of the beam. x-axis is parallel to the paper and the y-axis is perpendicular to the paper.

A beam object can be propagated through a free space or made to interact with an optics.

As a beam propagate through optical system, optical distance and Gouy phase are accumerated.

=== Attributes ===

q: q-parameter of the beam. If the beam is eliptic, q is the q-parameter of the best matching circular mode.

qx: q-parameter of the beam in the x-direction. [complex float]

qy: q-parameter of the beam in the y-direction. [complex float]

pos: Position of the beam origin. [(2,) float array]

dirVect: Propagation direction vector. [(2,) float array]

dirAngle: Propagation direction angle measured from the positive x-axis. [float]

length: Length of the beam (used for DXF export) [float]

layer: Layer name of the beam when exported to a DXF file.

name: Name of the beam

wl: Wavelength in vacuum. Not the wavelength in the medium. [float]

n: Index of refraction of the medium the beam is passing through. [float]

P: Power [float]

wx: Beamwidth in x-direction [float]

wy: Beamwidth in y-direction [float]

optDist: Accumulated optical distance.

Gouyx: Accumulated Gouy phase in x-direction.

Gouyy: Accumulated Gouy phase in y-direction.

Mx: ABCD matrix in x-direction. This is a 2x2 matrix representing the product of ABCD transformations applied to this beam. It defaults to an identity matrix. Whenever a beam experience an ABCD matrix transformation, such as propagation in the space or reflection by a curved mirror, the applied ABCD matrix is multiplied to this matrix, so that we can keep track of what kind of transformations were made during beam propagation.

My: ABCD matrix in y-direction. The meaning is the same as Mx.

departSurfAngle: The angle formed by x-axis and the normal vector of the surface from which the beam is departing. Default is None. Used by the drawing routine.

departSurfInvROC: Inverse of the ROC of the surface from which the beam is departing. The ROC is positive for a concave surface seen from the beam side. Default is None. Used by the drawing routine.

incSurfAngle: The angle formed by the x-arm and the normal vector of the surface to which the beam is incident. Default is None. Used by the drawing routine.

incSurfInvROC: Inverse of the ROC of the surface to which the beam is incident. The ROC is positive for a concave surface seen from the beam side. Default is None. Used by the drawing routine.

stray_order: An integer indicating if this beam is a stray light or not. The default value is 0. Every time a beam is reflected by an AR surface or transmits an HR surface, this couter is increased by 1.

ABCDTrans(*ABCDx*, *ABCDy=None*)
Apply ABCD transformation to the beam.

ABCDx: ABCD matrix for x-direction

ABCDy: ABCD matrix for y-direction

R(*dist=0.0*)
Returns the beam ROC at a distance dist from the origin of the beam.

copy()
Make a deep copy.

draw(*cv*, *sigma=3.0*, *mode='x'*, *drawWidth=True*, *fontSize=False*, *drawPower=False*, *drawROC=False*, *drawGouy=False*, *drawOptDist=False*, *drawName=False*, *debug=False*)
Draw the beam into a DXF object.

Arguments

cv: draw.Canvas object.

sigma: The width of the beam drawn is sigma * (1/e^2 radius of the beam). The default is sigma = 3. sigma = 2.7 gives 1ppm diffraction loss. [float]

mode: 'avg', 'x', or 'y'. A beam can have different widths for x- and y- directions. If 'avg' is specified, the average of them are drawn. 'x' and 'y' specifies to show the width of the respective directions.

fontSize: Size of the font used to show supplemental informations. [float]

drawWidth: Whether to draw width or not. [Boolean]

drawPower: Whether to show the beam power. [Boolean]

drawROC: Whether to show the ROC or not. [Boolean]

drawGouy: Whether to show the Gouy phase or not. [Boolean]

drawOptDist: Whether to show the accumulated optical distance or not. [Boolean]

drawName: Whether draw the name of the beam or not. [Boolean]

drawWidth(*cv*, *sigma*, *mode*)

drawWidthOld(*dxf*, *sigma*, *mode*)

flip(*flipDirVect=True*)

Change the propagation direction of the beam by 180 degrees. This is equivalent to the reflection of the beam by a spherical mirror with the same ROC as the beam.

If optional argument flipDirVect is set to False, the propagation direction of the beam is not changed.

propagate(*d*)

Propagate the beam by a distance d from the current position. self.n is used as the index of refraction. During this process, the optical distance traveled is added to self.optDist. self.Goux and self.Gouyy are also updated to record the Gouy phase change.

rotate(*angle*, *center=False*)

Rotate the beam around 'center'. If center is not given, the beam is rotated around self.pos.

Arguments

angle: Rotation angle in radians.

center: Center for rotation. [(2,) array of float]

translate(*trVect*)

Translate the beam by the direction and the distance specified by a vector.

Arguments

trVect: A vector to specify the translation direction and distance. [(2,) float array]

waist()

Return the tuples of waist size and distance

width(*dist*)

Returns the beam width at a distance dist from the origin of the beam. The width is the readius where the light power becomes 1/e^2.

gtrace.beam.**optFunForEndPointR**(*phi*, *Mrot*, *R*, *q0*, *k*, *sigma*, *side*)

A function to return the distance between the point on the spherical surface at an angle phi and the beam width at the same z.

gtrace.beam.**optFunForFlat**(*a*, *Mrot*, *q0*, *k*, *sigma*, *side*)

A function to return the distance between the point on the spherical surface at an angle phi and the beam width at the same z.

gtrace.beam.**optFunForStartPointR**(*phi*, *Mrot*, *R*, *q0*, *k*, *sigma*, *side*)
A function to return the distance between the point on the spherical surface at an angle phi and the beam width at the same z.

gtrace.beam.**optimCrossPointFlat**(*theta*, *q0*, *k*, *sigma*)

gtrace.beam.**optimEndPointR**(*theta*, *R*, *q0*, *k*, *sigma*)

gtrace.beam.**optimStartPointR**(*theta*, *R*, *q0*, *k*, *sigma*)

3.1.3 hello Module

class gtrace.hello.**hello**

greetings()

gtrace.hello.**saysomething**()

3.1.4 nonsequential Module

gtrace.nonsequential

A module to perform non-sequential trace of a beam in an optical system.

gtrace.nonsequential.**non_seq_trace**(*optList*, *src_beam*, *order=10*, *power_threshold=0.1*)
Perform non-sequential trace of the source beam, src_beam, through the optical system represented by a collection of optics, optList.

The return value of this function is a list of beams.

3.1.5 optcomp Module

Define optical components for gtrace.

class gtrace.optcomp.**Mirror**(*HRcenter=[0.0, 0.0], normAngleHR=0.0, normVectHR=None, diameter=0.25, thickness=0.15, wedgeAngle=0.004363323129985824, inv_ROC_HR=0.00014285714285714287, inv_ROC_AR=0.0, Refl_HR=0.99, Trans_HR=0.01, Refl_AR=0.01, Trans_AR=0.99, n=1.45, name='Mirror', HRtransmissive=False, term_on_HR=False*)
Bases: gtrace.optcomp.Optics

Representing a partial reflective mirror.

=== Attributes ===

HRcenter: The position of the center of the arc of the HR surface. [(2,) float array]

HRcenterC: The position of the center of the chord of the HR surface. [(2,) float array]

normVectHR: Normal vector of the HR surface.

normAngleHR: Angle of the HR normal vector.

ARcenter: The position of the center of the AR surface. [(2,) float array]

normVectAR: Normal vector of the HR surface.

normAngleAR: Angle of the HR normal vector.

HRtransmissive: A boolean value defaults to False. If True, this mirror is supposed to transmit beams on the HR surface. Therefore, for the first encounter of a beam on the HR surface of this mirror will not increase the stray_order. This flag should be set to True for beam splitters and input test masses.

term_on_HR: If this is True, a beam with stray_order=0 will be terminated when it hits on HR. This is to avoid the inifinite loop of non-sequencial trace by forming a cavity.

copy()

draw(*cv*, *drawName=False*)
Draw itself

hit(*beam*, *order=0*, *threshold=0.0*, *face=False*)
A function to hit the optics with a beam.

This function attempts to hit the optics with the source beam, `beam`.

Input parameters:

`beam`: A GaussianBeam object to be interacted by the optics.

`order`: An integer to specify how many times the internal reflections are computed.

`threshold` The power threshold for internal reflection calculation. If the power of an auxiliary beam falls below this threshold, further propagation of this beam will not be performed.

Return values `(isHit, beamDict, face)`

`isHit` This is a boolean to answer whether the beam hit the optics or not.

`beamDict` A dictionary containing resultant beams.

face: An optional string identifying which face of the optics was hit. For a mirror, `face` is any of "HR", "AR" or "side".

hitFromAR(*beam*, *order=0*, *threshold=0.0*, *verbose=False*)
Compute the reflected and deflected beams when an input beam hit the AR surface.

The internal reflections are computed as long as the number of internal reflections are below the `order` and the power of the reflected beams is over the threshold.

hitFromHR(*beam*, *order=0*, *threshold=0.0*, *verbose=False*)
Compute the reflected and deflected beams when an input beam hit the HR surface.

The internal reflections are computed as long as the number of internal reflections are below the `order` and the power of the reflected beams is over the threshold.

isHit(*beam*)
A function to see if a beam hits this optics or not.

Input parameters

`beam`: A GaussianBeam object to be interacted by the optics.

Returned value

The return value is a dictionary with the following keys: `isHit, position, distance, face`

`isHit`: This is a boolean to answer whether the beam hit the optics or not.

`position`: A numpy array containing the coordinate values of the intersection point between the beam and the optics. If isHit is False, this parameter does not mean anything.

`distance` The distance between the beam origin and the intersection point.

face: An optional string identifying which face of the optics was hit. For example, face can be either "HR" or "AR" for a mirror. face can also be "side", meaning that the beam hits a side of the optics, which is not meant to be used, e.g. the side of a mirror. In this case, the beam have reached a dead end.

rotate(*angle*, *center=False*)
Rotate the mirror. If center is not specified, the center of rotation is HRcenter. If center is given (as a vector), the center of rotation is center. center is a position vector in the global coordinates.

translate(*trVect*)

class gtrace.optcomp.**Optics**
Bases: traits.has_traits.HasTraits

A general optics class from which other specific optics classes are derived.

=== Attributes === name: Name of the optics. [string] center: Center position of the optics. [(2,) float array] rotationAngle: This angle defines the orientation of the optics.

hit(*beam*, *order=0*, *threshold=0.0*)
A function to hit the optics with a beam.

This function attempts to hit the optics with the source beam, beam.

Input parameters:

beam: A GaussianBeam object to be interacted by the optics.

order: An integer to specify how many times the internal reflections are computed.

threshold The power threshold for internal reflection calculation. If the power of an auxiliary beam falls below this threshold, further propagation of this beam will not be performed.

Return values (isHit, beamDict, face)

isHit This is a boolean to answer whether the beam hit the optics or not.

beamDict A dictionary containing resultant beams.

face: An optional string identifying which face of the optics was hit. For a mirror, face is any of "HR", "AR" or "side".

isHit(*beam*)
A function to see if a beam hits this optics or not.

Input parameters

beam: A GaussianBeam object to be interacted by the optics.

Returned value

The return value is a dictionary with the following keys: isHit, position, distance, face

isHit: This is a boolean to answer whether the beam hit the optics or not.

position: A numpy array containing the coordinate values of the intersection point between the beam and the optics. If isHit is False, this parameter does not mean anything.

distance The distance between the beam origin and the intersection point.

face: An optional string identifying which face of the optics was hit. For example, face can be either "HR" or "AR" for a mirror. face can also be "side", meaning that the beam hits a side of the optics, which is not meant to be used, e.g. the side of a mirror. In this case, the beam have reached a dead end.

3.1.6 unit Module

gtrace.unit.**deg2rad**(*deg*)

gtrace.unit.**rad2deg**(*rad*)

3.1.7 Subpackages

draw Package

draw Package

draw Module

Drawing classes for gtrace

class gtrace.draw.draw.**Arc**(*center*, *radius*, *startangle*, *stopangle*, *thickness=0*, *angle_in_rad=True*)
Bases: gtrace.draw.draw.Shape

An arc

Note that angles are stored in rad.

class gtrace.draw.draw.**Canvas**(*unit='m'*)
Bases: object

Canvas class

add_layer(*name*, *color=(0, 0, 0)*)

add_shape(*shape*, *layername*)

class gtrace.draw.draw.**Circle**(*center*, *radius*, *thickness=0*)
Bases: gtrace.draw.draw.Shape

A circle

class gtrace.draw.draw.**Layer**(*name*, *color=(0, 0, 0)*)
Bases: object

Layer class

add_shape(*shape*)

class gtrace.draw.draw.**Line**(*start*, *stop*, *thickness=0*)
Bases: gtrace.draw.draw.Shape

Line class

exception gtrace.draw.draw.**NumberOfElementError**
Bases: exceptions.BaseException

class gtrace.draw.draw.**PolyLine**(*x*, *y*, *thickness=0*)
Bases: gtrace.draw.draw.Shape

A light weight poly-line

class gtrace.draw.draw.**Rectangle**(*point*, *width*, *height*, *thickness=0*)
Bases: gtrace.draw.draw.Shape

A rectangle

class gtrace.draw.draw.**Shape**
Bases: object

Shape class

class gtrace.draw.draw.**Text**(*text*, *point*, *height=1.0*, *rotation=0.0*, *angle_in_rad=True*)
Bases: gtrace.draw.draw.Shape

Text

Note that angles are stored in rad.

dxf Module

dxf.py - a DXF export library for python

Sample code:

import dxf d = dxf.DXF('test.dxf') d.add_layer('ABC', color=5) d.add_entity(dxf.Line((1.5,5), (56,-89)), 'ABC') d.save_to_file()

class gtrace.draw.dxf.**Arc**(*center*, *radius*, *startangle*, *stopangle*, *thickness=0*)
Bases: gtrace.draw.dxf.Entity

An arc entity

draw(*layername*)

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

class gtrace.draw.dxf.**Circle**(*center*, *radius*, *thickness=0*)
Bases: gtrace.draw.dxf.Entity

A circle entity

draw(*layername*)

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

class gtrace.draw.dxf.**DXF**(*filename='drawing.dxf'*)
Bases: object

A DXF file class.

add_entity(*entity*, *layername*)

add_layer(*name*, *color=1*, *ltype='Continuous'*)

save_to_file()
Save the DXF file

class gtrace.draw.dxf.**Entity**
Bases: object

A graphic entity

draw()

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

set_handle(*handle*)

class gtrace.draw.dxf.**Layer**(*name*, *handle*, *color=1*, *ltype='Continuous'*)
Bases: object

Layer class

add_entity(*entity*)

class gtrace.draw.dxf.**Line**(*start*, *stop*, *thickness=0*)
Bases: gtrace.draw.dxf.Entity

A line entity

draw(*layername*)

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

class gtrace.draw.dxf.**LwPolyLine**(*x*, *y*, *thickness=0*)
Bases: gtrace.draw.dxf.Entity

A light weight poly-line

draw(*layername*)

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

exception gtrace.draw.dxf.**NumberOfElementError**
Bases: exceptions.BaseException

class gtrace.draw.dxf.**Rectangle**(*point*, *width*, *height*, *thickness=0*)
Bases: gtrace.draw.dxf.Entity

A rectangle

draw(*layername*)

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

class gtrace.draw.dxf.**Text**(*text*, *point*, *height=1.0*, *rotation=0.0*)
Bases: gtrace.draw.dxf.Entity

Text

draw(*layername*)

report_min_max()
Return the coordinates of the lower left and the upper right corners of the drawing. Return value: ((xmin, ymin), (xmax, ymax))

gtrace.draw.dxf.**color_encode**(*color*)
Given a set of RGB values for a color, find the closest matching one from the pre-defined colors in the DXF specification. Then return its color code (an integer in 1 to 255).

= Input = color: A tuple of three numbers in the range of 0-255, i.e. (R,G,B)

= Return = best_color_num: integer

gtrace.draw.dxf.**test_func**()

renderer Module

Renderer module for gtrace.draw

exception gtrace.draw.renderer.**UnknownShapeError**
Bases: exceptions.BaseException

gtrace.draw.renderer.**renderDXF**(*canvas*, *filename*)
Render a canvas into a DXF file

tools Module

gtrace.draw.tools.**drawAllBeams**(*d*, *beamList*, *sigma=3.0*, *drawWidth=True*, *drawPower=False*, *drawROC=False*, *drawGouy=False*, *drawOptDist=False*, *layer=None*, *mode='x'*, *fontSize=0.01*)

gtrace.draw.tools.**drawAllOptics**(*d*, *opticsList*, *drawName=True*, *layer=None*)

gtrace.draw.tools.**drawOptSys**(*optList*, *beamList*, *filename*, *fontSize=False*)

gtrace.draw.tools.**rotateAll**(*objList*, *angle*, *center*)

gtrace.draw.tools.**transAll**(*objList*, *transVect*)

optics Package

cavity Module

cavity.py - A Cavity class and related functions for representing a Fabry-Perot cavity

class gtrace.optics.cavity.**Cavity**(*r1=0.9*, *r2=0.99*, *L=1.0*, *R1=-1.5*, *R2=1.5*, *wl=1.064e-06*, *power=False*)
Bases: traits.has_traits.HasTraits

A class to represent a Fabry-Perot cavity.

Attributes

r1: Input mirror reflectivity (amplitude)

r2: End mirror reflectivity (amplitude)

rp1: Input mirror reflectivity (power)

rp2: End mirror reflectivity (power)

L: Length

R1: ROC of the input mirror (positive when concave to incident light, i.e. convex seen from inside the cavity)

R2: ROC of the end mirror (positive when concave to incident light, i.e. concave seen from inside the cavity)

wl: Wavelength

FSR()
Returns the free spectral range of the cavity.

Nbounce()
Bounce number

finesse()
Returns the finesse of the cavity.

intra(*f=0*, *d=0*)
Returns the intra cavity field amplitude. It assumes the cavity was locked to the incident light first. Then computes the intra-cavity field amplitude for the light with a frequency shift f from the original light with the cavity length changed by d from the initial state.

== *Input arguments* ==

f: Frequency shift of the light in Hz.

d: Cavity length detuning in m.

== *Returned parameter* ==

The intra-cavity field amplitude at the input mirror surface (a complex number).

modeSpacing()
Return the transverse mode spacing of the cavity (commonly called gamma). It is a fractional number defined by gamma = (mode spacing frequency)/FSR.

pole()
Cavity pole frequency [Hz]

powerGain()
Ratio of the intra-cavity power to the input power.

refl(*f=0*, *d=0*)
Returns the amplitude reflectivity of the cavity. It assumes the cavity was locked to the incident light first. Then computes the amplitude reflectivity for the light with a frequency shift f from the original light with the cavity length changed by d from the initial state.

== *Input arguments* ==

f: Frequency shift of the light in Hz.

d: Cavity length detuning in m.

== *Returned parameter* ==

The amplitude reflectivity of the cavity (a complex number).

spotSize()
Returns the beam spot sizes on the input and end mirrors as a tuple (w1,w2).

storageTime()
Storage time

trans(*f=0*, *d=0*)
Returns the amplitude transmissivity of the cavity. It assumes the cavity was locked to the incident light first. Then computes the amplitude transmissivity for the light with a frequency shift f from the original light with the cavity length changed by d from the initial state.

== *Input arguments* ==

f: Frequency shift of the light in Hz.

d: Cavity length detuning in m.

== *Returned parameter* ==

The amplitude transmissivity of the cavity (a complex number).

waist (*size=False*)
Return the q-parameter or the radius of the beam at the cavity waist.

Input arguments

size: (optional) if set to true, the first element of the returned tuple will be the waist size, rather than the q-parameter.

==Returned parameters ==

(q0, d): This function returns a tuple with two elements. The first element is the q-parameter of the cavity mode at the cavity waist. If size=True is given, it becomes the waist size (1/e^2 radius). The second element is the distance of the cavity waist from the input mirror.

gtrace.optics.cavity.**finesse**(*r1*, *r2*, *power=False*)
Returns the finesse of a cavity

consts Module

gtrace.optics.consts.**n_fused_silica**(*wl*)
Calculate the index of refraction of fused silica for a given wavelength.

gtrace.optics.consts.**n_sapphire_extraordinary**(*wl*)
Calculate the index of refraction of Sapphire extraordinary axis for a given wavelength.

gtrace.optics.consts.**n_sapphire_ordinary**(*wl*)
Calculate the index of refraction of Sapphire ordinary axis for a given wavelength.

gtrace.optics.consts.**sellmeier**(*wl*, *B1*, *B2*, *B3*, *C1*, *C2*, *C3*)
Calculate index of refraction using Sellmeiers equation

n^2 = 1+B1*wl^2/(wl^2 - C1) + B2*wl^2/(wl^2 - C2) + B3*wl^2/(wl^2 - C3)

See below for the coefficients for specific materials. http://www.cvimellesgriot.com/products/Documents/Catalog/Dispersion_Equ

gaussian Module

gaussian - Gaussian Optics Module

This module contains several utility functions for gaussian optics.

gtrace.optics.gaussian.**ROCandWtoQ**(*ROC=1.0*, *w=1.0*, *wl=1.064e-06*)

gtrace.optics.gaussian.**Rw2q**(*ROC=1.0*, *w=1.0*, *wl=1.064e-06*)
Get the q-parameter from the ROC and w.

gtrace.optics.gaussian.**appertureCut**(*r=1.0*, *w=3.0*)

gtrace.optics.gaussian.**beamClip**(*a=1.0*, *w=3.0*)

gtrace.optics.gaussian.**modeMatching**(*q1*, *q2x*, *q2y=False*)
Mode matching between two beams with different q-parameters. The axes of the two beams are assumed to be matched.

q1: q-parameter of the first beam. This beam is assumed to be circular.

q2x: q-parameter of the second beam in x-direction. If the second beam is also circular, omit the next argument.

q2y: q-parameter of the second beam in y-direction. Specify this parameter if the second beam is eliptic.

gtrace.optics.gaussian.**modeSpacing**(*g1*, *g2*)

gtrace.optics.gaussian.**optimalMatching**(*q1*, *q2*)
Returns a mode (q-parameter) which best matches the given two q-parameters, q1 and q2.

Returned values: (q, match)

q: The best matching q-parameter

match: Mode matching rate

gtrace.optics.gaussian.**q2R**(*q*)
Convert a q-parameter to the ROC

gtrace.optics.gaussian.**q2w**(*q*, *wl=1.064e-06*)
Convert a q-parameter to the beam size

gtrace.optics.gaussian.**q2zr**(*q*)
Convert a q-parameter to Rayleigh range.

gtrace.optics.gaussian.**qToROC**(*q*)

gtrace.optics.gaussian.**qToRadius**(*q*, *wl=1.064e-06*)

gtrace.optics.gaussian.**w02zr**(*w0*, *wl=1.064e-06*)
Convert Rayleigh range to the waist size

gtrace.optics.gaussian.**zr2w0**(*zr*, *wl=1.064e-06*)
Convert Rayleigh range to the waist size

geometric Module

gtrace.optics.geometric.**deflection_angle**(*theta*, *n1*, *n2*, *deg=True*)

gtrace.optics.geometric.**line_arc_intersection**(*pos*, *dirVect*, *chord_center*, *chordNormVect*, *invROC*, *diameter*, *verbose=False*)
Compute the intersection point between a line and an arc.

pos: Origin of the line dirVect: Direction of the line chord_center: The center of the chord made by the arc. chordNormVect: Normal vector of the chord. invROC: Inverse of the ROC of the arc. Positive for concave surface. diameter: Length of the chord.

gtrace.optics.geometric.**line_plane_intersection**(*pos*, *dirVect*, *plane_center*, *normalVector*, *diameter*)
Compute the intersection point between a line and a plane

A line is specified by its origin (pos) and the direction vector (dirVect). A plane is specfied by its center coordinates (plane_center) and the normal vector (normalVector). The plane has its size (diameter).

The returned value is a dictionary of with the following keys: "Intersection Point": numpy array of the coordinates of the intersection point. "isHit": A boolean value of whether the line intersects with the plane or not. "distance": Distance between the origin of the line and the intersection point. "distance from center": Distance between the center of the plane and the intersection point.

gtrace.optics.geometric.**normSpheric**(*normAngle*, *invROC*, *dist_from_center*)
Returns the local normal angle of a spheric mirror at a distance from the center.

normAngle: The angle formed by the normal vector of the mirror at the center and the x-axis.

invROC: 1/R, where R is the ROC of the mirror.

dist_from_center: The distance from the center of the point where the local normal is requested. This is a signed value. For a mirror facing +x (the normal vector points towards positive x direction), this distance is positive for points with positive y coordinate, and negative for points with negative y coordinate.

gtrace.optics.geometric.**refl_defl_angle**(*beamAngle*, *normAngle*, *n1*, *n2*, *invROC=None*)
Returns a tuples of reflection and deflection angles.

beamAngle: The angle formed by the propagation direction vector of the incident beam and the x-axis.

normAngle: The angle formed by the normal vector of the surface and the x-axis.

n1: Index of refraction of the incident side medium.

n2: Index of refraction of the transmission side medium.

gtrace.optics.geometric.**vc_deflect**(*theta*, *theta1*, *n1*, *n2*)
Deflection angle helper function for VariCAD. theta is the angle of the surface measured from right. theta1 is the angle of the incident beam measured from right. It returns an angle of the deflected beam measured from right.

gtrace.optics.geometric.**vc_reflect**(*theta*, *theta1*)

gtrace.optics.geometric.**vector_rotation_2D**(*vect*, *angle*)

unit Module

gtrace.optics.unit.**deg2rad**(*deg*)

gtrace.optics.unit.**rad2deg**(*rad*)

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

g

gtrace.__init__, 7
gtrace.beam, 7
gtrace.draw, 13
gtrace.draw.draw, 13
gtrace.draw.dxf, 14
gtrace.draw.renderer, 16
gtrace.draw.tools, 16
gtrace.hello, 10
gtrace.nonsequential, 10
gtrace.optcomp, 10
gtrace.optics.cavity, 16
gtrace.optics.consts, 18
gtrace.optics.gaussian, 18
gtrace.optics.geometric, 19
gtrace.optics.unit, 20
gtrace.unit, 13

INDEX

A

ABCDTrans() (gtrace.beam.GaussianBeam method), 8
add_entity() (gtrace.draw.dxf.DXF method), 14
add_entity() (gtrace.draw.dxf.Layer method), 15
add_layer() (gtrace.draw.draw.Canvas method), 13
add_layer() (gtrace.draw.dxf.DXF method), 14
add_shape() (gtrace.draw.draw.Canvas method), 13
add_shape() (gtrace.draw.draw.Layer method), 13
appertureCut() (in module gtrace.optics.gaussian), 18
Arc (class in gtrace.draw.draw), 13
Arc (class in gtrace.draw.dxf), 14

B

beamClip() (in module gtrace.optics.gaussian), 18

C

Canvas (class in gtrace.draw.draw), 13
Cavity (class in gtrace.optics.cavity), 16
Circle (class in gtrace.draw.draw), 13
Circle (class in gtrace.draw.dxf), 14
color_encode() (in module gtrace.draw.dxf), 15
copy() (gtrace.beam.GaussianBeam method), 8
copy() (gtrace.optcomp.Mirror method), 11

D

deflection_angle() (in module gtrace.optics.geometric), 19
deg2rad() (in module gtrace.optics.unit), 20
deg2rad() (in module gtrace.unit), 13
draw() (gtrace.beam.GaussianBeam method), 8
draw() (gtrace.draw.dxf.Arc method), 14
draw() (gtrace.draw.dxf.Circle method), 14
draw() (gtrace.draw.dxf.Entity method), 14
draw() (gtrace.draw.dxf.Line method), 15
draw() (gtrace.draw.dxf.LwPolyLine method), 15
draw() (gtrace.draw.dxf.Rectangle method), 15
draw() (gtrace.draw.dxf.Text method), 15
draw() (gtrace.optcomp.Mirror method), 11
drawAllBeams() (in module gtrace.draw.tools), 16
drawAllOptics() (in module gtrace.draw.tools), 16
drawOptSys() (in module gtrace.draw.tools), 16
drawWidth() (gtrace.beam.GaussianBeam method), 9
drawWidthOld() (gtrace.beam.GaussianBeam method), 9
DXF (class in gtrace.draw.dxf), 14

E

Entity (class in gtrace.draw.dxf), 14

F

finesse() (gtrace.optics.cavity.Cavity method), 16
finesse() (in module gtrace.optics.cavity), 18
flip() (gtrace.beam.GaussianBeam method), 9
FSR() (gtrace.optics.cavity.Cavity method), 16

G

GaussianBeam (class in gtrace.beam), 7
greetings() (gtrace.hello.hello method), 10
gtrace.__init__ (module), 7
gtrace.beam (module), 7
gtrace.draw (module), 13
gtrace.draw.draw (module), 13
gtrace.draw.dxf (module), 14
gtrace.draw.renderer (module), 16
gtrace.draw.tools (module), 16
gtrace.hello (module), 10
gtrace.nonsequential (module), 10
gtrace.optcomp (module), 10
gtrace.optics.cavity (module), 16
gtrace.optics.consts (module), 18
gtrace.optics.gaussian (module), 18
gtrace.optics.geometric (module), 19
gtrace.optics.unit (module), 20
gtrace.unit (module), 13

H

hello (class in gtrace.hello), 10
hit() (gtrace.optcomp.Mirror method), 11
hit() (gtrace.optcomp.Optics method), 12
hitFromAR() (gtrace.optcomp.Mirror method), 11
hitFromHR() (gtrace.optcomp.Mirror method), 11

I

intra() (gtrace.optics.cavity.Cavity method), 17

isHit() (gtrace.optcomp.Mirror method), 11
isHit() (gtrace.optcomp.Optics method), 12

L

Layer (class in gtrace.draw.draw), 13
Layer (class in gtrace.draw.dxf), 15
Line (class in gtrace.draw.draw), 13
Line (class in gtrace.draw.dxf), 15
line_arc_intersection() (in module gtrace.optics.geometric), 19
line_plane_intersection() (in module gtrace.optics.geometric), 19
LwPolyLine (class in gtrace.draw.dxf), 15

M

Mirror (class in gtrace.optcomp), 10
modeMatching() (in module gtrace.optics.gaussian), 18
modeSpacing() (gtrace.optics.cavity.Cavity method), 17
modeSpacing() (in module gtrace.optics.gaussian), 18

N

n_fused_silica() (in module gtrace.optics.consts), 18
n_sapphire_extraordinary() (in module gtrace.optics.consts), 18
n_sapphire_ordinary() (in module gtrace.optics.consts), 18
Nbounce() (gtrace.optics.cavity.Cavity method), 16
non_seq_trace() (in module gtrace.nonsequential), 10
normSpheric() (in module gtrace.optics.geometric), 19
NumberOfElementError, 13, 15

O

optFunForEndPointR() (in module gtrace.beam), 9
optFunForFlat() (in module gtrace.beam), 9
optFunForStartPointR() (in module gtrace.beam), 9
Optics (class in gtrace.optcomp), 12
optimalMatching() (in module gtrace.optics.gaussian), 18
optimCrossPointFlat() (in module gtrace.beam), 10
optimEndPointR() (in module gtrace.beam), 10
optimStartPointR() (in module gtrace.beam), 10

P

pole() (gtrace.optics.cavity.Cavity method), 17
PolyLine (class in gtrace.draw.draw), 13
powerGain() (gtrace.optics.cavity.Cavity method), 17
propagate() (gtrace.beam.GaussianBeam method), 9

Q

q2R() (in module gtrace.optics.gaussian), 19
q2w() (in module gtrace.optics.gaussian), 19
q2zr() (in module gtrace.optics.gaussian), 19
qToRadius() (in module gtrace.optics.gaussian), 19
qToROC() (in module gtrace.optics.gaussian), 19

R

R() (gtrace.beam.GaussianBeam method), 8
rad2deg() (in module gtrace.optics.unit), 20
rad2deg() (in module gtrace.unit), 13
Rectangle (class in gtrace.draw.draw), 13
Rectangle (class in gtrace.draw.dxf), 15
refl() (gtrace.optics.cavity.Cavity method), 17
refl_defl_angle() (in module gtrace.optics.geometric), 19
renderDXF() (in module gtrace.draw.renderer), 16
report_min_max() (gtrace.draw.dxf.Arc method), 14
report_min_max() (gtrace.draw.dxf.Circle method), 14
report_min_max() (gtrace.draw.dxf.Entity method), 14
report_min_max() (gtrace.draw.dxf.Line method), 15
report_min_max() (gtrace.draw.dxf.LwPolyLine method), 15
report_min_max() (gtrace.draw.dxf.Rectangle method), 15
report_min_max() (gtrace.draw.dxf.Text method), 15
ROCandWtoQ() (in module gtrace.optics.gaussian), 18
rotate() (gtrace.beam.GaussianBeam method), 9
rotate() (gtrace.optcomp.Mirror method), 12
rotateAll() (in module gtrace.draw.tools), 16
Rw2q() (in module gtrace.optics.gaussian), 18

S

save_to_file() (gtrace.draw.dxf.DXF method), 14
saysomething() (in module gtrace.hello), 10
sellmeier() (in module gtrace.optics.consts), 18
set_handle() (gtrace.draw.dxf.Entity method), 15
Shape (class in gtrace.draw.draw), 13
spotSize() (gtrace.optics.cavity.Cavity method), 17
storageTime() (gtrace.optics.cavity.Cavity method), 17

T

test_func() (in module gtrace.draw.dxf), 15
Text (class in gtrace.draw.draw), 14
Text (class in gtrace.draw.dxf), 15
trans() (gtrace.optics.cavity.Cavity method), 17
transAll() (in module gtrace.draw.tools), 16
translate() (gtrace.beam.GaussianBeam method), 9
translate() (gtrace.optcomp.Mirror method), 12

U

UnknownShapeError, 16

V

vc_deflect() (in module gtrace.optics.geometric), 20
vc_reflect() (in module gtrace.optics.geometric), 20
vector_rotation_2D() (in module gtrace.optics.geometric), 20

W

w02zr() (in module gtrace.optics.gaussian), 19

waist() (gtrace.beam.GaussianBeam method), 9
waist() (gtrace.optics.cavity.Cavity method), 17
width() (gtrace.beam.GaussianBeam method), 9

Z

zr2w0() (in module gtrace.optics.gaussian), 19