""" Superposition Eye Pathlength and Absorption Program

Original QBASIC version by Magnus L Johnson and Genevre Parker, 1995 Python rewrite by Stephen P Moss, 2012-2013

<http://about.me/gawbul> [gawbul@gmail.com](mailto:gawbul@gmail.com)

"""

author

**=** "Steve Moss"

copyright

**=** "Copyright 1995-2013, Magnus L Johnson and Stephen P Moss"

credits

license

version

**= [**"Steve Moss"**,** "Magnus Johnson"**,** "Genevre Parker"**]**

**=** "GPLv3"

**=** "0.49b"

maintainer

**=** "Steve Moss"

email

**=** ["gawbul@gmail.com"](mailto:gawbul@gmail.com)

status

**=** "beta"

# import modules

**import** os**,** sys**,** time**,** re # needed for os, system, time and regular expression specific functions

**from** datetime **import** timedelta**,** date # needed for time specific functions **import** math # needed for math functions (self.pi, cos, sin, tan, atan) **import** getopt # needed to get options from command line

**import** rpy2 # needed for plotting subroutines in R

#import pygame # needed for graphics output \*\*\* not yet implemented \*\*\*

# main handler subroutine

**def** main**():**

"""Controls the main program flow."""

# check what the program arguments are and assign appropriate variables

opts\_array **=** handle\_options**(**sys**.**argv**[**1**:]) (**input\_file**,** graphicsopt**) =** opts\_array

# check whether the user provide an input filename

**if** input\_file**:**

# process file process\_input\_file**(**input\_file**,** graphicsopt**)** sys**.**exit**()**

# else:

# just continue with inline parameters below

# pass

# show startup information

startup**()**

# track how long it takes

start **=** time**.**time**()**

# if not using an input file for the parameters you can set them manually as follows

# setup nephrops\_eye as new SuperpositionEye object - with relevant parameters passed

# using Nephrops norvegicus flat lateral measurments

# see README file or GitHub for information on parameters

**print** "Setting up new superposition eye object..."

nephrops\_eye **=** SuperpositionEye**(**"nephrops"**,** 180**,** 25**,** 7800**,** 50**,** 3200**,** 1.34**,** 1.37**,** 18**,** 0**)**

# run the model

**print** "Running the ray tracing model (please wait)..."

nephrops\_eye**.**run\_model**(**graphicsopt**)**

# summarise the data

**print** "Outputting summary data..."

nephrops\_eye**.**summarise\_data**()**

# how long did we take?

end **=** time**.**time**()** took **=** end **-** start

**print** "\nFinished in %s seconds.\n" **%** timedelta**(**seconds**=**took**)**

# handle any program input options given at the command line

**def** handle\_options**(**optsargs**):**

"""Handles the input arguments to the program."""

# process using getopts

# try:

**(**opts**,** args**) =** getopt**.**getopt**(**optsargs**,** "f:gchv"**, [**"file="**,** "graphics"**,** "citation"**,** "help"**,** "version"**])**

**except** getopt**.**GetoptError **as** err**:**

**print** str**(**err**)** usage**()** sys**.**exit**(**2**)**

filename **= None** graphicsopt **= False for** o**,** a **in** opts**:**

**if** o **in (**"-f"**,** "--file"**):**

filename **=** a

**elif**o **in (**"-g"**,** "--graphics"**):** graphicsopt **= True** sys**.**exit**()**

**elif** o **in (**"-c"**,** "--citation"**):**

startup**()** sys**.**exit**()**

**elif** o **in (**"-h"**,** "--help"**):**

usage**()** sys**.**exit**()**

**elif** o **in (**"-v"**,** "--version"**):**

version **=** version

**print** "pathlen.py version %s" **%** version sys**.**exit**()**

# else:

**assert False,** "unhandled option"

**return** filename**,** graphicsopt

# display startup information in the terminal

**def** startup**():**

"""Displays information about the program on startup, or via the citation input argument."""

**print** "\nPathLength - Implements a ray tracing model to calculate resolution and sensitivity in reflective superposition compound eyes."

**print** "-" **\*** len**(**"PathLength - Implements a ray tracing model to calculate resolution and sensitivity in reflective superposition compound eyes."**)**

**print** "If you use this program, please cite:"

**print** "\nGaten, E., Moss, S., Johnson, M. 2013. The Reniform Reflecting Superposition Compound Eyes of Nephrops Norvegicus: Optics, \n" \

"Susceptibility to Light-Induced Damage, Electrophysiology and a Ray Tracing Model. In:

M. L. Johnson and M. P. Johnson, ed(s).\n" \

"Advances in Marine Biology: The Ecology and Biology of Nephrops norvegicus. Oxford:

Academic Press, 107:148."

**print** "-" **\*** len**(**"Susceptibility to Light-Induced Damage, Electrophysiology and a Ray Tracing Model. In: M. L. Johnson and M. P. Johnson, ed(s)."**) +** "\n"

# return

# display usage information to the terminal

**def** usage**():**

"""Displays usage information via the help input argument."""

**print** "The valid program options are:"

**print** "\t-f or --file\t\tAllows the user to provide a csv input file with sets\n\t\t\t\tof parameters for individual runs on individual lines."

**print** "\t-g or --graphics\tTurn graphics on or off. \*\*\* not yet implemented \*\*\*"

**print** "\t-c or --citation\tDisplays the citation information." **print** "\t-h or --help\t\tDisplays this usage information." **print** "\t-v or --version\t\tDisplays the program version."

**print** "\n\tFor more information visit https://github.com/gawbul/pathlength/\n\tor email Steve Moss (gawbul@gmail.com)."

# return

# process the parameter input file

**def** process\_input\_file**(**filename**,** graphicsflag**):**

"""Processes a csv input file for multiple parameter model testing."""

# first check the file exists and exist with error message if not

**if not** os**.**path**.**exists**(**filename**):**

**print** "Error: Filename \'%s\' does not exist" **%** filename sys**.**exit**()**

# track how long it takes

start **=** time**.**time**()**

# file must exist, so open and parse inputfile **=** open**(**filename**,** "r"**)** count **=** 1

**for** line **in** inputfile**.**readlines**():**

# check if it is a comment line

# filter out whitespace and any dodgy characters and split into parts

parts **=** re**.**sub**(**'\s\W'**,** ''**,** line**).**split**(**","**)**

# check we have the right number of parameters

**if** len**(**parts**) ==** 0**:**

# this just means that there was a blank line?

# continue

**if** len**(**parts**) !=** 10**:**

**print** "Error: The number of parameters is incorrect on line %d." **%** count

# continue

# assign the variables from the parts list

**(**sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**) =** parts

# create an object

**print** "Setting up new superposition eye object..."

eye\_object\_from\_file **=** SuperpositionEye**(**str**(**sn**),** int**(**rl**),** float**(**rw**),** int**(**ed**),** float**(** fw**),** int**(**ad**),** float**(**cri**),** float**(**rri**),** int**(**bce**),** float**(**pra**))**

# run the model

**print** "Running the ray tracing model (please wait)..."

eye\_object\_from\_file**.**run\_model**(**graphicsflag**)**

# summarise the data

**print** "Outputting summary data..."

eye\_object\_from\_file**.**summarise\_data**()**

# increment line count

count **+=** 1

# how long did we take?

end **=** time**.**time**()** took **=** end **-** start

**print** "\nFinished in %s seconds.\n" **%** timedelta**(**seconds**=**took**)**

# return

# setup superposition eye class

# class SuperpositionEye():

**def** init **(**self**,** sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**):**

"""Initialises the default variables of a new SuperpositionEye object."""

# store parameters incase needed in future

self**.**eye\_parameters **= [**sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**]**

# set variables for output data self**.**rowdata **= []** self**.**output\_data **= []**

self**.**aa **=** 100**\*[**0**]**

self**.**ab **=** 100**\*[**0**]**

# set variables for calculations

self**.**pi **=** math**.**pi # define self.pi

self**.**conv **=** self**.**pi **/** 180 # convert radians to degrees (1 degree = self.pi / 180 radians)

self**.**proximal\_rhabdom\_angle **=** pra # used for pointy rhabdoms

# set files for output data

self**.**setup\_files**(**sn**)** # passes species name to prepend output filenames

self**.**iteration\_count **=** 1 # q = 0 in original

self**.**shielding\_pigment\_length **=** 0.0 # extent of shielding pigment set to zero

# check the blur circle extent isn't set to less than 1 otherwise we will get division by zero error

**if** bce **<** 1**:**

bce **=** 1

self**.**blur\_circle\_extent **=** bce # blur circle extent

# input data - eye parameters

self**.**rhabdom\_length **=** float**(**rl**)** # rhabdom length

self**.**increment\_amount **=** self**.**rhabdom\_length **/** 10 # amount to increment tapetum or pigment

self**.**reflective\_tapetum\_length **=** 0.0 # extent of tapetal pigment set to zero

self**.**num\_facets **=** 0 # num of facets across aperture self**.**rhabdom\_width **=** rw # rhabdom width/diameter self**.**aperture\_diameter **=** ad # aperture diameter

self**.**y **=** 0 # y??? - set to one originally, but we use 0 based indexing in python

self**.**facet\_width **=** fw # facet width

self**.**eye\_diameter **=** ed # eye diameter

# undeclared in original code self**.**boa **=** 0 # boa??? self**.**tot **=** 0 # tot???

self**.**col\_total **=** 0 # total rhaboms?

self**.**row\_total **=** 0 # total facets?

**def** initial\_calculations**(**self**):**

"""Does some initial calculations before running the main model."""

# do initial calculations

**(**sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**) =** self**.**eye\_parameters # get stored parameters

self**.**eye\_circumference **=** self**.**pi **\*** self**.**eye\_diameter # circumference of eye self**.**aperture\_radius **=** self**.**aperture\_diameter **/** 2 # aa in original code - aperture radius

self**.**eye\_radius **=** self**.**eye\_diameter **/** 2 # eye radius

self**.**da **=** math**.**sqrt**((**self**.**eye\_radius **\*\*** 2**) - (**self**.**aperture\_radius **\*\*** 2**))** # DA??? self**.**ac **=** math**.**atan**(**self**.**aperture\_radius **/** self**.**da**) /** self**.**conv # AC??? self**.**aperture\_diameter **= (**self**.**ac **/** 360**) \*** self**.**eye\_circumference # change aperture diameter

self**.**optical\_axis **= (**self**.**facet\_width **/** self**.**eye\_circumference**) \*** 360 # calculate optical axis from eye circumference and facet width

self**.**facet\_num **=** 1 # facet number

self**.**num\_facets **=** int**(**self**.**aperture\_diameter **/** self**.**facet\_width**)** # num of facets across aperture

self**.**rhabdom\_radius **=** self**.**rhabdom\_width **/** 2 # rhabdom radius self**.**old\_rhabdom\_length **=** self**.**rhabdom\_length # old rhabdom length self**.**max\_rhabdom\_length **=** self**.**rhabdom\_length # store rhabdom length for main loop self**.**inter\_ommatidial\_angle **=** 0 # inter-ommatidial angle

self**.**current\_facet **=** 0 # current facet

# angle of total internal reflection (rhabdoms) self**.**cytoplasm\_ri **=** cri # cytoplasm refractive index self**.**rhabdom\_ri **=** rri # rhabdom refractive index

self**.**snells\_law **=** math**.**asin**(**self**.**cytoplasm\_ri **/** self**.**rhabdom\_ri**) /** self**.**conv # calculate angle for total internal reflection using Snell's law self**.**critical\_angle **=** 90 **-** self**.**snells\_law # critical angle below which light is totally internally reflected within rhabdom

self**.**mx **=** math**.**sqrt**((**self**.**rhabdom\_length **\*\*** 2**) + (**self**.**rhabdom\_radius **\*\*** 2**))** # mx???

# output initial P (pigment) and T (tapetum) to output file one self**.**write\_output**(**self**.**outputfile\_one**,** self**.**shielding\_pigment\_length**)** # write pigment length to output file

self**.**write\_output**(**self**.**outputfile\_one**,** self**.**reflective\_tapetum\_length**)** # write tapetum length to output file

self**.**cz **=** 0 # increases angle of acceptance of rhabdom - initialise to false

# return

**def** run\_model**(**self**,** graphicsflag**):**

"""Main workhorse of the program. Runs the ray tracing model with the given parameters."""

# print start\_time and write to debug file

start\_time **=** time**.**time**()**

self**.**write\_output**(**self**.**debug\_file**,** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n%s\n" **%** date**.**fromtimestamp**(** start\_time**).**strftime**(**"%d/%m/%Y %H:%M:%S"**))**

# do the initial calculations

self**.**initial\_calculations**()**

# main program loop

# while True:

# calculate prox-dist length of first pass

**if** self**.**boa **>** self**.**critical\_angle **and** self**.**boa **<** 25 **and** self**.**cz **==** 0**:**

# change shape of proximal portion of the rhabdom

self**.**boa **-=** self**.**proximal\_rhabdom\_angle

**if** self**.**inter\_ommatidial\_angle **==** 0**:**

# ray absorbed by proximal shielding pigment

self**.**case\_four**()**

# else:

self**.**y **=** self**.**rhabdom\_radius **/** math**.**tan**(**self**.**boa **\*** self**.**conv**)**

**if** self**.**y **>=** self**.**rhabdom\_length**:**

# ray reflected off base of rhabdom by tapetum

self**.**case\_three**()**

**elif** self**.**y **>** self**.**rhabdom\_length **-** self**.**shielding\_pigment\_length **or** self**.**y **>**

self**.**rhabdom\_length **-** self**.**reflective\_tapetum\_length **or** self**.**boa **<** self**.** critical\_angle**:**

self**.**case\_two**()**

# else:

# light passes through rhabdom

self**.**case\_one**()**

# goto 1002

**if** self**.**rhabdom\_length **<=** self**.**reflective\_tapetum\_length **or** self**.** rhabdom\_length **<=** self**.**shielding\_pigment\_length**:**

# pass else:

# \*\*\* call display graphics here \*\*\*

# continue

self**.**current\_facet **+=** 1

self**.**rhabdom\_radius **=** self**.**rhabdom\_width **/** 2 self**.**rhabdom\_length **=** self**.**old\_rhabdom\_length self**.**inter\_ommatidial\_angle **+=** self**.**optical\_axis self**.**cz **=** 0 # set CZ as false

# row complete append 998 and output to file self**.**col\_total **=** len**(**self**.**rowdata**)** self**.**rowdata**.**append**(**998**)** self**.**write\_output**(**self**.**outputfile\_one**,** self**.**rowdata**)**

# append row to output\_data for outputfile\_two self**.**output\_data**.**append**(**self**.**rowdata**[**0**:-**1**])** self**.**row\_total **=** len**(**self**.**output\_data**)**

# clear self.rowdata

self**.**rowdata **= []**

# account for refraction at cornea

**if** self**.**inter\_ommatidial\_angle **<** 60**:**

self**.**boa **= (**self**.**inter\_ommatidial\_angle **\*** 0.8677**) +** 3.38

**if** self**.**inter\_ommatidial\_angle **<** 50**:**

self**.**boa **= (**self**.**inter\_ommatidial\_angle **\*** 0.9196**) +** 0.8676

**if** self**.**inter\_ommatidial\_angle **<** 35**:**

self**.**boa **= (**self**.**inter\_ommatidial\_angle **\*** 0.9407**) +** 0.1648

**if** self**.**inter\_ommatidial\_angle **<** 15**:**

self**.**boa **= (**self**.**inter\_ommatidial\_angle **\*** 0.9494**) +** 0.004667

**if** self**.**inter\_ommatidial\_angle **>** 60**:**

self**.**print\_output**(**"\*\*\* UNREAL ANGLE AT CORNEA \*\*\*"**)**

self**.**write\_output**(**self**.**outputfile\_one**,** "UNREAL ANGLE AT CORNEA"**)**

# light loss at cone due to angle of incidence

self**.**cc **=** self**.**facet\_width **/** math**.**tan**(**self**.**boa **\*** self**.**conv**)** # CC???

**if** self**.**cc **> (**self**.**facet\_width **\*** 2**):**

self**.**fw **=** math**.**cos**(**self**.**inter\_ommatidial\_angle **\*** self**.**conv**) \*** self**.** facet\_width # FW???

# else:

self**.**ll **= ((**2 **\*** self**.**cc**) - (**2 **\*** self**.**facet\_width**))**

self**.**fw **=** math**.**sin**(**self**.**inter\_ommatidial\_angle **\*** self**.**conv**) \*** self**.**ll self**.**facet\_num **=** self**.**fw **/** self**.**facet\_width

# account for change in angle between adjacent rhabdoms

self**.**fd **=** self**.**num\_facets **/** self**.**blur\_circle\_extent # FD??? - this is used to divide the aperture up

# if current facet is outside edge of eyeshine patch then break out of for loop

# otherwise check where the current facet is and transfer POL to appropriate rhabdom accordingly

self**.**nx **=** 1

**for** i **in** range**(**self**.**blur\_circle\_extent**):**

**if** self**.**current\_facet **>=** self**.**num\_facets**:**

# pass

**elif** self**.**current\_facet **>= (**self**.**fd **\*** self**.**nx**):** self**.**boa **+=** self**.**optical\_axis self**.**rowdata**.**append**(**0**)**

self**.**nx **+=** 1

# check to see if edge of eyeshine patch has been reached

**if** self**.**current\_facet **>=** self**.**num\_facets**:**

# pass else:

**continue**

# iterate over output data

**for** col **in** range**(**self**.**col\_total**):**

**for** row **in** range**(**self**.**row\_total**):**

**if** self**.**col\_total **>** len**(**self**.**output\_data**[**row**]):**

**for** i **in** range**(**self**.**col\_total **-** len**(**self**.**output\_data**[**row**])):** self**.**output\_data**[**row**].**append**(**0**)**

# check all rows

**if** self**.**output\_data**[**row**][**col**] >** 0**:**

self**.**ab**[**col**] =** 1 **-** math**.**exp**(-**0.0067 **\*** self**.**output\_data**[**row**][**col**])**

**elif** self**.**output\_data**[**row**][**col**] ==** 0**:**

self**.**ab**[**col**] =** 0

**if** col **==** 0 **and** self**.**ab**[**col**] >** 0**:**

self**.**bx **=** 100 **\*** self**.**ab**[**col**]**

**if** col **>** 0 **and** self**.**ab**[**col**] >** 0**:**

self**.**bx **=** 100 **\* ((**1 **-** self**.**tot**) \*** self**.**ab**[**col**])**

**if** self**.**ab**[**col**] ==** 0**:**

self**.**bx **=** 0

self**.**tot **+= (**self**.**bx **/** 100**)** self**.**aa**[**col**] +=** self**.**bx self**.**bx **=** 0

self**.**write\_output**(**self**.**debug\_file**, [**col **+** 1**,** self**.**aa**[**col**],** self**.**ab**[**col**],** row **+** 1**])**

self**.**bx **=** 0

self**.**tot **=** 0

self**.**x **=** 0

output\_tmp **= []** output\_tmp**.**append**(**self**.**reflective\_tapetum\_length**)** output\_tmp**.**append**(**self**.**shielding\_pigment\_length**) for** i **in** range**(**self**.**col\_total**):**

self**.**aa**[**i**] =** int**(**self**.**aa**[**i**] /** self**.**row\_total**)** output\_tmp**.**append**(**self**.**aa**[**i**])**

self**.**aa**[**i**] =** 0 output\_tmp**.**append**(**999**)** self**.**print\_output**(**""**)**

self**.**write\_output**(**self**.**outputfile\_two**,** output\_tmp**)** self**.**bx **=** 0

# reset tapetum to zero and increase pigment by 10%

**if** self**.**reflective\_tapetum\_length **>=** self**.**max\_rhabdom\_length **and** self**.** shielding\_pigment\_length **>=** self**.**max\_rhabdom\_length**:**

# increment iteration count and output count to screen and 999 to the file

self**.**iteration\_count **+=** 1

self**.**write\_output**(**self**.**outputfile\_one**,** 999**)**

# end of program sys**.**stdout**.**write**(**"\a"**)** # beep sys**.**stdout**.**flush**()** # flush beep **break**

**elif** self**.**reflective\_tapetum\_length **>=** self**.**max\_rhabdom\_length **and** self**.** shielding\_pigment\_length **<** self**.**max\_rhabdom\_length**:**

self**.**reflective\_tapetum\_length **=** 0

self**.**shielding\_pigment\_length **+=** self**.**increment\_amount

# else:

self**.**reflective\_tapetum\_length **+=** self**.**increment\_amount

# increment iteration count and output count to screen and 999 to the file

self**.**iteration\_count **+=** 1

self**.**write\_output**(**self**.**outputfile\_one**,** 999**)**

# reset output data

self**.**output\_data **= []**

# reset parameters

self**.**reset\_parameters**()**

# print end\_time

end\_time **=** time**.**time**()**

self**.**write\_output**(**self**.**debug\_file**,** "\n%s\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" **%** date**.**fromtimestamp**(** end\_time**).**strftime**(**"%d/%m/%Y %H:%M:%S"**))**

**def** case\_one**(**self**):**

# no reflection - light passes through rhabdom

self**.**x **=** self**.**rhabdom\_radius **/** math**.**sin**(**self**.**boa **\*** self**.**conv**)** self**.**rowdata**.**append**(**self**.**x **\*** self**.**facet\_num**)** self**.**rhabdom\_length **-=** self**.**y

self**.**boa **+=** self**.**optical\_axis self**.**cz **=** 1 # set CZ to true

**def** case\_two**(**self**):**

# reflection from edge

self**.**x **=** self**.**rhabdom\_radius **/** math**.**sin**(**self**.**boa **\*** self**.**conv**)**

self**.**z **= (**self**.**rhabdom\_length **-** self**.**y**) /** math**.**cos**(**self**.**boa **\*** self**.**conv**)**

**if** self**.**z **>** self**.**x**:** self**.**z **=** self**.**x

**if (**self**.**x **+** self**.**z**) >** self**.**old\_rhabdom\_length**:** self**.**v **=** self**.**x **+** self**.**z

**elif(**self**.**x **+** self**.**z**) <** self**.**old\_rhabdom\_length**:** self**.**v **=** self**.**old\_rhabdom\_length

**if** self**.**reflective\_tapetum\_length **==** 0**:**

val **= (**self**.**x **+** self**.**z**) \*** self**.**facet\_num

**elif** self**.**reflective\_tapetum\_length **>** 0**:**

val **= (**self**.**x **+** self**.**z **+** self**.**v**) \*** self**.**facet\_num

**if** self**.**shielding\_pigment\_length **>** 0**:**

val **= (**self**.**x **+** self**.**z**) \*** self**.**facet\_num

**if** self**.**shielding\_pigment\_length **> (**self**.**rhabdom\_length **-** self**.**y**):** val **=** self**.**x **\*** self**.**facet\_num

self**.**rowdata**.**append**(**val**)**

# return

**def** case\_three**(**self**):**

# bounce off base

**if** self**.**y **==** self**.**rhabdom\_length**:** self**.**x **=** self**.**mx

**if** self**.**y **>** self**.**rhabdom\_length**:**

self**.**x **=** self**.**rhabdom\_length **/** math**.**cos**(**self**.**boa **\*** self**.**conv**)**

**if** self**.**x **>** self**.**old\_rhabdom\_length**:** self**.**v **=** self**.**x

**if** self**.**x **<** self**.**old\_rhabdom\_length**:** self**.**v **=** self**.**old\_rhabdom\_length

**if** self**.**reflective\_tapetum\_length **==** 0**:**

val **=** self**.**x **\*** self**.**facet\_num

**if** self**.**reflective\_tapetum\_length **>** 0**:**

val **= (**self**.**x **+** self**.**v**) \*** self**.**facet\_num

**if** self**.**shielding\_pigment\_length **>** 0**:**

val **=** self**.**x **\*** self**.**facet\_num self**.**rowdata**.**append**(**val**)**

# return

**def** case\_four**(**self**):**

# perpendicular ray

**if** self**.**reflective\_tapetum\_length **>** 0**:**

val **= (**self**.**rhabdom\_length **\*** 2**) \*** self**.**facet\_num

**if** self**.**reflective\_tapetum\_length **==** 0**:**

val **=** self**.**rhabdom\_length **\*** self**.**facet\_num

**if** self**.**shielding\_pigment\_length **>** 0**:**

val **=** self**.**rhabdom\_length **\*** self**.**facet\_num

self**.**rowdata**.**append**(**val**)**

# return

**def** setup\_files**(**self**,** sn**):**

"""Setup the filenames and remove old ones if they exist."""

# get current directory and build filenames species\_name **=** sn**.**lower**()** # always convert to lowercase curr\_dir **=** os**.**getcwd**()** # get current working directory

self**.**outputfile\_one **=** os**.**path**.**join**(**curr\_dir**,** species\_name **+** '\_output\_one.csv'**)** # outputfile one

self**.**outputfile\_two **=** os**.**path**.**join**(**curr\_dir**,** species\_name **+** '\_output\_two.csv'**)** # outputfile two

self**.**matrixfile\_one **=** os**.**path**.**join**(**curr\_dir**,** species\_name **+** '\_summary\_one.csv'**)** # matrixfile one

self**.**matrixfile\_two **=** os**.**path**.**join**(**curr\_dir**,** species\_name **+** '\_summary\_res.csv'**)** # matrixfile two

self**.**matrixfile\_three **=** os**.**path**.**join**(**curr\_dir**,** species\_name **+** '\_summary\_sen.csv'**)** # matrixfile three

self**.**debug\_file **=** os**.**path**.**join**(**curr\_dir**,** species\_name **+** '\_debug.txt'**)** # debug file

# check if files exist and delete them

**if** os**.**path**.**exists**(**self**.**outputfile\_one**):** os**.**remove**(**self**.**outputfile\_one**)**

**if** os**.**path**.**exists**(**self**.**outputfile\_two**):** os**.**remove**(**self**.**outputfile\_two**)**

**if** os**.**path**.**exists**(**self**.**matrixfile\_one**):** os**.**remove**(**self**.**matrixfile\_one**)**

**if** os**.**path**.**exists**(**self**.**matrixfile\_two**):** os**.**remove**(**self**.**matrixfile\_two**)**

**if** os**.**path**.**exists**(**self**.**matrixfile\_three**):** os**.**remove**(**self**.**matrixfile\_three**)**

**if** os**.**path**.**exists**(**self**.**debug\_file**):** os**.**remove**(**self**.**debug\_file**)**

# return

**def** write\_output**(**self**,** filename**,** data**):**

"""Write data to an output filename."""

# open file for append and write data

filehandle **=** open**(**filename**,** 'a'**)** # open file in append mode

**if** isinstance**(**data**,** list**):**

csv\_data **=** ","**.**join**(**map**(**str**,** data**))**

# else:

csv\_data **=** str**(**data**)**

filehandle**.**write**(**csv\_data **+** "\n"**)** # write output\_text string to file with new line character

filehandle**.**close**()** # close file

# return

**def** print\_output**(**self**,** text**):**

"""Output text and progress information to the screen.""" **print** "%d: (T:%0.2f P:%0.2f) %s" **% (**self**.**iteration\_count**,** self**.** reflective\_tapetum\_length**,** self**.**shielding\_pigment\_length**,** text**) return**

**def** reset\_parameters**(**self**):**

"""Reset all the parameters to their default values."""

# get stored parameters

**(**sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**) =** self**.**eye\_parameters

# reset eye parameters using stored values self**.**num\_facets **=** 0 # num of facets across aperture self**.**rhabdom\_width **=** rw # rhabdom width/diameter self**.**aperture\_diameter **=** ad # aperture diameter

self**.**y **=** 0 # y??? - set to one originally, but we use 0 based indexing in python

self**.**facet\_width **=** fw # facet width

self**.**eye\_diameter **=** ed # eye diameter

# do the initial calculations self**.**initial\_calculations**() return**

**def** return\_parameters**(**self**):**

"""Get the original parameters, as stored at the beginning of the program."""

# get stored parameters

**(**sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**) =** self**.**eye\_parameters

# return parameters to user

**return** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce

**def** summarise\_data**(**self**):**

"""Summarise the data produced by the calculations in the run\_model function."""

# get stored parameters

**(**sn**,** rl**,** rw**,** ed**,** fw**,** ad**,** cri**,** rri**,** bce**,** pra**) =** self**.**eye\_parameters

# set required parameters self**.**facet\_width **=** fw self**.**eye\_diameter **=** ed

self**.**eye\_circumference **= (**22.0 **/** 7.0**) \*** float**(**self**.**eye\_diameter**)** # need 22.0 / 7.0 here as rounds down to 3 with being an integer

self**.**inter\_ommatidial\_angle **= (**self**.**facet\_width **/** self**.**eye\_circumference**) \*** float**(**360**)** self**.**reflective\_tapetum\_length **=** 0

self**.**shielding\_pigment\_length **=** 0

self**.**absorbance **=** 0

self**.**facet **=** 0

self**.**rhabdom **=** 0 self**.**rhabdoms **=** 21**\*[**0**]** self**.**tot **=** 0

self**.**bx **=** 0

self**.**torus **=** 0

self**.**inci **=** 0

self**.**area **=** 0

self**.**arem **=** 0

self**.**sens **=** 0

self**.**rhab **=** 0

self**.**rens **=** 0

self**.**cc **=** 0

self**.**dd **=** 0

self**.**frac **=** 0

self**.**oab **=** 0 self**.**matrix\_sens **= []** self**.**matrix\_rhab **= []** self**.**matrix\_res **= []**

# setup outputfile filehandle

filehandle **=** open**(**self**.**outputfile\_one**,** 'r'**)**

# iterate over file

**for** line **in** filehandle**.**readlines**():** line **=** line**.**rstrip**()**

**if not** line**:**

# break

**if** re**.**match**(**"^([0-9]+\.[0-9]{1,})$"**,** line**):**

**if** self**.**reflective\_tapetum\_length **==** 0**:**

self**.**reflective\_tapetum\_length **=** float**(**line**)**

**elif** self**.**shielding\_pigment\_length **==** 0**:**

self**.**shielding\_pigment\_length **=** float**(**line**)**

**elif** re**.**match**(**"^([0-9\.\,\s]+998)$"**,** line**) and** line **!=** "999"**:**

text **=** re**.**sub**(**"\s+"**,** ""**,** line**)** parts **=** text**.**split**(**','**)**

**for** part **in** parts**:**

# check if end of line

**if** part **==** "998"**:**

self**.**rhabdom **=** 0

self**.**bx **=** 0

self**.**tot **=** 0

self**.**facet **+=** 1

self**.**area **=** self**.**pi **\* (**self**.**facet **+** 0.5**) \*\*** 2

**if** self**.**facet **==** 0**:**

self**.**torus **=** self**.**pi **\* (**0.5**) \*\*** 2 self**.**inci **=** self**.**pi **\* (**self**.**facet **-** 0.5**) \*\*** 2 self**.**torus **=** self**.**area **-** self**.**inci

**if** self**.**area **>** self**.**arem**:** self**.**arem **=** self**.**area

# else:

part **=** float**(**part**)** # convert to float for calculations

**if** part **>** 0**:**

self**.**absorbance **=** 1 **-** math**.**exp**(-**0.01 **\*** part**)** # calculate absorbance

# else:

self**.**absorbance **=** 0 # light doesn't strike rhabdom

**if** self**.**rhabdom **==** 0 **and** self**.**absorbance **>** 0**:**

self**.**bx **= (**100 **\*** self**.**absorbance**)** # axial rhabdom

**elif** self**.**rhabdom **>** 0 **and** self**.**absorbance **>** 0**:**

self**.**bx **= (**100 **\* ((**1 **-** self**.**tot**) \*** self**.**absorbance**))**

**if** self**.**absorbance **==** 0**:**

self**.**bx **=** 0 # bx = light not absorbed

self**.**tot **+= (**self**.**bx **/** 100**)**

self**.**bx **\*=** self**.**torus

**for** i **in** range**(**len**(**self**.**rhabdoms**)):**

**if** self**.**rhabdom **==** i**:** self**.**rhabdoms**[**i**] +=** self**.**bx

self**.**rhabdom **+=** 1 # increment rhabdom

self**.**bx **=** 0

**elif** line **==** "999"**:**

# finished block of numbers - work out absorption

self**.**rhabdom **=** 0

self**.**sens **=** sum**(**self**.**rhabdoms**)** self**.**rhab **=** self**.**rhabdoms**[**0**] /** self**.**sens

self**.**halfway\_point **=** self**.**rhabdoms**[**0**] /** 2

self**.**xz **=** self**.**rhabdoms**[**0**]** self**.**yy **=** self**.**rhabdoms**[**1**]**

self**.**optic\_axis **=** 0

**for** i **in** range**(**1**,** 12**):**

**if** self**.**halfway\_point **<** self**.**rhabdoms**[**i**]:** self**.**xz **=** self**.**rhabdoms**[**i**]**

self**.**yy **=** self**.**rhabdoms**[**i**+**1**]**

self**.**optic\_axis **=** self**.**inter\_ommatidial\_angle **\*** i self**.**diff **=** self**.**xz **-** self**.**yy

self**.**hwp **=** self**.**xz **-** self**.**halfway\_point self**.**frac **=** self**.**hwp **/ (**self**.**diff **+** 0.1**)**

self**.**oab **=** self**.**frac **\*** self**.**inter\_ommatidial\_angle self**.**res **=** self**.**oab **+** self**.**optic\_axis # width at 50% point **for** i **in** range**(**16**):**

self**.**rhabdoms**[**i**] =** int**(**self**.**rhabdoms**[**i**])**

**if** self**.**cc **==** 0**:** self**.**matrix\_sens**.**append**(**0**)** self**.**matrix\_rhab**.**append**(**0**)** self**.**matrix\_res**.**append**(**0**)**

self**.**write\_output**(**self**.**matrixfile\_three**,** self**.**matrix\_sens**)** self**.**write\_output**(**self**.**matrixfile\_one**,** self**.**matrix\_rhab**)** self**.**write\_output**(**self**.**matrixfile\_two**,** self**.**matrix\_res**)** self**.**matrix\_sens **= []**

self**.**matrix\_rhab **= []**

self**.**matrix\_res **= []** self**.**matrix\_sens**.**append**(**int**(**self**.**sens **/** self**.**arem**))** self**.**matrix\_rhab**.**append**(**int**(**self**.**rhab **\*** 100**))**

self**.**matrix\_res**.**append**(**int**(**self**.**res **\*** 200**))** self**.**print\_output**(**"CC: %s DD: %s" **% (**str**(**self**.**cc**),** str**(**self**.**dd**)))** self**.**iteration\_count **+=** 1

self**.**cc **+=** 1

**if** self**.**cc **==** 11**:**

self**.**dd **+=** 1

self**.**cc **=** 0

self**.**rhabdoms**[**0**] =** 0

self**.**rhabdoms**[-**1**] =** 0

self**.**bx **=** 0

self**.**facet **=** 0

self**.**reflective\_tapetum\_length **=** 0

self**.**shielding\_pigment\_length **=** 0 self**.**write\_output**(**self**.**matrixfile\_three**,** self**.**matrix\_sens**)** self**.**write\_output**(**self**.**matrixfile\_one**,** self**.**matrix\_rhab**)** self**.**write\_output**(**self**.**matrixfile\_two**,** self**.**matrix\_res**)** self**.**matrix\_sens **= []**

self**.**matrix\_rhab **= []**

self**.**matrix\_res **= []**

# close filehandle

filehandle**.**close**()**

# let user know we've finished

# end of program sys**.**stdout**.**write**(**"\a"**)** # beep sys**.**stdout**.**flush**()** # flush beep

self**.**print\_output**(**"\*\*\* End of program \*\*\*"**)**

# return

**def** build\_plots**(**self**):**

"""This function will produce publication quality plots from the output data."""

# return

# check for main subroutine and call it

**if** name

**==** " main "**:**

sys**.**exit**(**main**())**