LuminetCpp

Generated by Doxygen 1.10.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

angular_properties . . . ??
BHphysics . . . ??
BlackHole . . . ??
cloud_points . . . ??
meshes::compare . . . ??
CSVRow . . . ??
meshes::driehoek . . . ??
find_redshift_params . . . ??
ir_params . . . ??
irs_solver_params . . . ??
Isolines . . . ??
Isoradial . . . ??
IsoRedShift . . . ??
meshes::Mesh . . . ??
meshes::MeshPoint . . . ??
OperatorsOrder2 . . . ??
plot_params . . . ??
Plotter . . . ??
meshes::Point . . . ??
meshes::Segment . . . ??
solver_params . . . ??
Source . . . ??
Tests . . . ??

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Include/BlackHole.h . . . . . ??
Include/BlackHolePhysics.h . . . . . ??
Include/isolines.h . . . . . ??
Include/IsoRadials.h . . . . . ??
Include/IsoRedShift.h . . . . . ??
Include/mesh.h . . . . . ??
Include/plotter.h . . . . . ??
Include/TensorCalculus.h . . . . . ??
Include/Tests.h . . . . . ??
Include/utilities.h . . . . . ??

Chapter 3

Class Documentation

3.1 angular_properties Struct Reference

Public Attributes

- double **start_angle** = 0.0
- double **end_angle** = M_PI
- unsigned **angular_precision** = 500
- bool **mirror** = true

The documentation for this struct was generated from the following file:

- Include/utilities.h

3.2 BHphysics Class Reference

Public Member Functions

- double **zeta_r** (double periastron, double r, double bh_mass)
- void **get_plot** (const std::vector< double > &, const std::vector< double > &, const std::vector< double > &, const double &)
- double **cos_alpha** (double phi, double incl)
- double **alpha** (double phi, double incl)
- std::vector< double > **filter_periastrons** (const std::vector< double > &periastron, double bh_mass, double tol)
- double **phi_inf** (double periastron, double M)
- double **mu** (double periastron, double bh_mass)

Static Public Member Functions

- static double **calc_q** (double periastron, double bh_mass, double)
- static double **calc_b_from_periastron** (double periastron, double bh_mass, double)
- static double **k** (double periastron, double bh_mass)
- static double **k2** (double periastron, double bh_mass, double)
- static double **zeta_inf** (double periastron, double bh_mass, double tol)
- static double **cos_gamma** (double _a, double incl, double tol)
- static double **eq13** (double periastron, double ir_radius, double ir_angle, double bh_mass, double incl, int n, double tol)
- static std::tuple< std::vector< double >, std::vector< double >, int > **midpoint_method** (const std::function< double(double, double, double, double, double, int, double)> func, const std::unordered_map< std::string, double > &args, const std::vector< double > &x, const std::vector< double > &y, int index_of_sign_change)
- static double **improve_solutions_midpoint** (const std::function< double(double, double, double, double, double, int, double)> &func, const std::unordered_map< std::string, double > &args, const std::vector< double > &x, const std::vector< double > &y, int index_of_sign_change, int iterations)
- static double **calc_periastron** (double _r, double incl, double _alpha, double bh_mass, int midpoint_iterations, bool plot_inbetween, int n, double min_periastron, int initial_guesses)
- static double **calc_impact_parameter** (double _r, double incl, double _alpha, double bh_mass, int midpoint_iterations, bool plot_inbetween, int n, double min_periastron, int initial_guesses, bool use_ellipse)
- static double **ellipse** (double r, double a, double incl)
- static double **flux_intrinsic** (double r, double acc, double bh_mass)
- static double **flux_observed** (double r, double acc, double bh_mass, double redshift_factor)
- static double **redshift_factor** (double radius, double angle, double incl, double bh_mass, double b_)
- static int **find_index_sign_change_indices** (const std::vector< double > &)
- static std::vector< double > **wavelengthToRGB** (const double &, const double &)
- static std::vector< double > **convert_TH** (const double &temperature, const double &brightness)
- static std::vector< double > **convert_NB** (const double &temperature, const double &brightness)

The documentation for this class was generated from the following files:

- Include/BlackHolePhysics.h
- Source/BlackHolePhysics.cpp

3.3 BlackHole Class Reference

Public Member Functions

- **BlackHole** (double mass=1.0, double inclination=80, double acc=1e-8)
- void **sample_Sources** (int n_points=1000, const std::string &f="", const std::string &f2="")
- void **calc_isoradials** (const std::vector< double > &direct_r, const std::vector< double > &ghost_r)
- void **add_isoradial** (Isoradial *isoradial, double radius, int order)
- std::map< double, IsoRedShift > **calc_isoredshifts** (std::vector< double > redshifts={ -0.15, 0.0, 0.1, 0.2, 0.5 }, const int &order=0)

The documentation for this class was generated from the following files:

- Include/BlackHole.h
- Source/BlackHole.cpp

3.4 cloud_points Struct Reference

Public Member Functions

- **cloud_points** (double x, double y, double redshift)

Public Attributes

- double **redshift_**
- double **x_**
- double **y_**

The documentation for this struct was generated from the following file:

- Include/IsoRedShift.h

3.5 meshes::compare Struct Reference

Public Member Functions

- bool **operator()** (std::size_t i, std::size_t j)

Public Attributes

- std::vector< double > const & **coords**
- double **cx**
- double **cy**

The documentation for this struct was generated from the following file:

- Include/mesh.h

3.6 CSVRow Class Reference

Public Member Functions

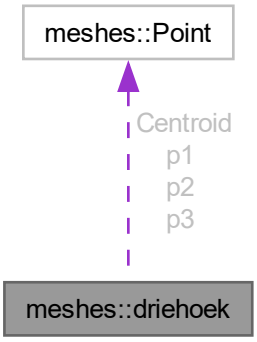
- std::string_view **operator[ ]** (std::size_t index) const
- std::size_t **size** () const
- void **readNextRow** (const std::string &line)

The documentation for this class was generated from the following file:

- Include/utilities.h

3.7 meshes::driehoek Struct Reference

Collaboration diagram for meshes::driehoek:



Public Member Functions

- **driehoek** (size_t ID_, Point p1_, Point p2_, Point p3_, double rs1_, double rs2_, double rs3_)
- bool **share_edge** (const driehoek &other) const
- std::set< Point > **shared_edge** (const driehoek &other) const

Public Attributes

- size_t **ID**
- size_t **ID1**
- size_t **ID2**
- size_t **ID3**
- size_t **nNeighbors**
- Point **p1**
- Point **p2**
- Point **p3**
- Point **Centroid**
- double **Rs1**
- double **Rs2**
- double **Rs3**

The documentation for this struct was generated from the following file:

- Include/mesh.h

3.8 find_redshift_params Struct Reference

Public Attributes

- bool **force_redshift_solution** = false
- unsigned **max_force_iter** = 5

The documentation for this struct was generated from the following file:

- Include/utilities.h

3.9 ir_params Struct Reference

Public Attributes

- double **start_angle** = 0.0
- double **end_angle** = M_PI
- unsigned **angular_precision** = 500
- bool **mirror** = true
- double **angular_margin** = 0.3

The documentation for this struct was generated from the following file:

- Include/utilities.h

3.10 irs_solver_params Struct Reference

Public Attributes

- unsigned **initial_guesses** = 12
- unsigned **midpoint_iterations** = 12
- double **times_inbetween** = 2
- unsigned **retry_angular_precision** = 15
- double **min_periastron** = 3.01
- bool **use_ellipse** = true
- unsigned **retry_tip** = 50
- unsigned **initial_radial_precision** = 15
- bool **plot_inbetween** = false
- double **angular_margin** = 0.3

The documentation for this struct was generated from the following file:

- Include/utilities.h

3.11 Isolines Class Reference

Public Member Functions

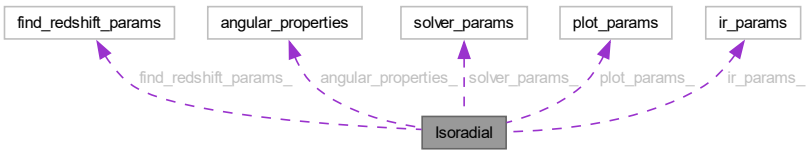
- **Isolines** (const std::shared_ptr< meshes::Mesh > &, const double &)
- std::vector< Point > **get_iso_lines** ()

The documentation for this class was generated from the following files:

- Include/isolines.h
- Source/isolines.cpp

3.12 Isoradial Class Reference

Collaboration diagram for Isoradial:



Public Member Functions

- **Isoradial** (double radius, double incl, double bh_mass, int order=0)
- **Isoradial** (double radius, double incl, double bh_mass, int order, angular_properties)
- std::pair< std::vector< double >, std::vector< double > > **get_bare_isoradials** ()
- void **calculate_ISCO_region** ()
- std::pair< std::vector< double >, std::vector< double > > **get_ISCO_region** ()
- std::pair< std::vector< double >, std::vector< double > > **get_ISCO_curve** ()
- std::vector< double > **get_redshift_factors** ()
- void **calculate** (const bool &isc=false)
- double **get_radius** ()
- std::pair< std::vector< double >, std::vector< double > > **calculate_coordinates** (const bool &isc=false)
- std::vector< double > **calc_redshift_factors** ()
- bool **isPointInsidePolygon** (const std::pair< double, double > &, const std::pair< std::vector< double >, std::vector< double > > &)
- std::vector< double > **find_angle** (double z)
- double **get_b_from_angle** (double angle)
- void **calc_between** (int ind)
- std::vector< double > **force_intersection** (double redshift)
- std::pair< std::vector< double >, std::vector< double > > **calc_redshift_location_on_ir** (double redshift, bool cartesian=false)

Public Attributes

- find_redshift_params **find_redshift_params_**
- angular_properties **angular_properties_**
- solver_params **solver_params_**
- plot_params **plot_params_**
- ir_params **ir_params_**
- std::vector< double > **X**
- std::vector< double > **Y**
- std::vector< double > **_radii_b**
- std::vector< double > **_angles**

The documentation for this class was generated from the following files:

- Include/IsoRadials.h
- Source/IsoRadials.cpp

3.13 IsoRedShift Class Reference

Public Member Functions

- **IsoRedShift** (const double &angle, const double &bh_mass, const double &lower_radius, const double &upper_radius, const size_t &_n_radii_, const size_t &_n_angles_, const double &)
- **IsoRedShift** (const double &, const double &, const double &, const std::map< double, std::pair< int, Isoradial * > > &)
- std::multimap< double, std::vector< meshes::Point > > **get_isolines** (const size_t n)
- std::pair< std::vector< double >, std::vector< double > > **get_ISCO_curve** ()
- std::pair< std::vector< double >, std::vector< double > > **get_ConcaveHull** ()
- void **improve** ()

Public Attributes

- double **redshift_**
- double **x_**
- double **y_**
- double **x_max** = -100000000000000.0
- double **x_min** = -x_max
- double **y_max** = x_max
- double **y_min** = x_min
- double **redshift_max** = -1000000000000.0
- double **redshift_min** = -redshift_max
- std::vector< double > **xIsco**
- std::vector< double > **yIsco**
- std::vector< meshes::Point > **ConcaveHull**
- std::vector< meshes::Point > **ISCO**
- double **delta** = 5.0e-2
- double **redshift_treshold**
- std::vector< double > **xCoordinates**
- std::vector< double > **yCoordinates**
- std::vector< double > **redshifts**
- std::vector< double > **xGrid**
- std::vector< double > **yGrid**
- std::vector< std::vector< double > > **redshiftGrid**

The documentation for this class was generated from the following files:

- Include/IsoRedShift.h
- Source/IsoRedShift.cpp

3.14 meshes::Mesh Class Reference

Public Member Functions

- **Mesh** (std::vector< double > const &, std::vector< double > const &, std::vector< double > const &, const std::vector< double > &, const std::vector< double > &yisco)
- **Mesh** (const Mesh &other)
- Mesh & **operator=** (const Mesh &other)
- std::vector< std::size_t > **getMesh** ()
- std::vector< double > **getCoords** ()
- void **triangulate** ()
- double **get_hull_area** ()
- std::vector< double > **get_hull_coords** ()
- std::vector< size_t > **get_hull_points** ()
- double **edge_length** (size_t e)
- size_t **get_interior_point** (size_t e)
- std::vector< double > **concavehull** (double chi_factor=0.1)
- std::vector< std::size_t > **get_triangles** ()
- std::vector< double > **get_tri_coordinates** ()
- std::size_t **legalize** (std::size_t a)
- std::size_t **hash_key** (double x, double y) const
- std::size_t **add_triangle** (std::size_t i0, std::size_t i1, std::size_t i2, std::size_t a, std::size_t b, std::size_t c)
- void **link** (std::size_t a, std::size_t b)
- size_t **next_halfedge** (size_t e)
- size_t **prev_halfedge** (size_t e)
- std::pair< Segment, Segment > **findConvexHullSegments** (const std::vector< double > &, const std::vector< double > &)
- bool **isOnPolygon** (const Point &, const Point &, const Point &)
- bool **oneIsOnPolygon** (const Point &)
- bool **is_left** (const Point &a, const Point &b, const Point &c)
- bool **isOutsideConcaveHull** (const Point &)
- void **makeISCO** ()
- void **make_driehoeken** ()

Public Attributes

- std::vector< double > **coords**
- std::vector< std::size_t > **triangles**
- std::vector< std::size_t > **halfedges**
- std::vector< std::size_t > **hull_prev**
- std::vector< std::size_t > **hull_next**
- std::vector< std::size_t > **hull_tri**
- std::size_t **hull_start**
- std::vector< std::size_t > **m_hash**
- double **m_center_x**
- double **m_center_y**
- std::size_t **m_hash_size**
- std::vector< std::size_t > **m_edge_stack**
- std::vector< double > **x_coords**
- std::vector< double > **y_coords**
- std::vector< double > **redshift_field**
- std::vector< driehoek > **driehoeken**
- std::vector< Point > **concaveHull**

- int **nConcaveHull**
- std::vector< double > **xISCO**
- std::vector< double > **yISCO**
- double **xIscoMax**
- double **xIscoMin**
- double **yIscoMax**
- double **yIscoMin**
- double **IscoEpsilon** =0.0
- std::vector< Point > **ISCO**

The documentation for this class was generated from the following files:

- Include/mesh.h
- Source/mesh.cpp

3.15 meshes::MeshPoint Struct Reference

Public Attributes

- std::size_t **i**
- double **x**
- double **y**
- std::size_t **t**
- std::size_t **prev**
- std::size_t **next**
- bool **removed**

The documentation for this struct was generated from the following file:

- Include/mesh.h

3.16 OperatorsOrder2 Class Reference

Public Member Functions

- **OperatorsOrder2** (int nGrid, double delta)
- std::vector< std::vector< std::vector< double > > > **laplace** (const std::vector< std::vector< std::vector< double > > > &fct)
- std::tuple< std::vector< std::vector< std::vector< double > > >, std::vector< std::vector< std::vector< double > > >, std::vector< std::vector< std::vector< double > > > > **gradient** (const std::vector< std::vector< std::vector< double > > > &fct)
- std::vector< std::vector< std::vector< double > > > **divergence** (const std::vector< std::vector< std::vector< double > > > &v_x, const std::vector< std::vector< std::vector< double > > > &v_y, const std::vector< std::vector< std::vector< double > > > &v_z)
- std::tuple< std::vector< std::vector< std::vector< double > > >, std::vector< std::vector< std::vector< double > > >, std::vector< std::vector< std::vector< double > > > > **gradDiv** (const std::vector< std::vector< std::vector< double > > > &A_x, const std::vector< std::vector< std::vector< double > > > &A_y, const std::vector< std::vector< std::vector< double > > > &A_z)
- std::tuple< double, double, double > **partialDerivs** (const std::vector< std::vector< std::vector< double > > > &fct, int i, int j, int k)
- std::vector< std::vector< std::vector< double > > > **Add** (const std::vector< std::vector< std::vector< double > > > &, const double &, const std::vector< std::vector< std::vector< double > > > &, const double &)

Static Public Member Functions

- static std::vector< double > **linspace** (const double &, const double &, const int &)
- static std::vector< double > **nonLinSpace** (const bool &, const double &, const int &)
- static double **phi** (const double &, const double &)
- static std::vector< double > **logspace** (const double &, const double &, const int &)
- static std::pair< std::vector< double >, std::vector< double > > **polar_to_cartesian_lists** (const std::vector< double > &radii, const std::vector< double > &angles, const double &rotation)
- static std::pair< double, double > **polar_to_cartesian_single** (double th, double radius, double rotation)
- static std::pair< double, double > **cartesian_to_polar** (double x, double y)
- static double **get_angle_around** (const std::vector< double > &p1, const std::vector< double > &p2)

The documentation for this class was generated from the following files:

- Include/TensorCalculus.h
- Source/TensorCalculus.cpp

3.17 plot_params Struct Reference

Public Attributes

- bool **plot_isoredshifts_inbetween** = false
- bool **save_plot** = false
- bool **plot_ellipse** = false
- bool **plot_core** = true
- bool **redshift** = true
- std::string **linestyle** = "-"
- double **linewidth** = 1.
- std::string **key** = ""
- std::string **face_color** = "black"
- std::string **line_color** = "white"
- std::string **text_color** = "white"
- double **alpha** = 1.
- bool **show_grid** = false
- bool **legend** = false
- bool **orig_background** = false
- bool **plot_disk_edges** = false
- std::pair< double, double > **ax_lim** = { -100, 100 }
- std::string **title** = "Isoradials for R ="

The documentation for this struct was generated from the following file:

- Include/utilities.h

3.18 Plotter Class Reference

Public Member Functions

- void **plot_isoradials** (std::vector< double > &, std::vector< double > &, std::vector< double > &, std::vector< double > &)
- void **plot_isoradials** (double, std::vector< double > &, std::vector< double > &, std::vector< double > &, std::vector< double > &, std::vector< double > &, bool=false)
- void **plot_iso_redshifts** (const double &, const std::multimap< double, std::vector< meshes::Point > > &, const double &, const double &, const double &, const double &, const std::pair< std::vector< double >, std::vector< double > > &, const std::pair< std::vector< double >, std::vector< double > > &, const bool &)
- void **plot_iso_redshifts** (const double &, const std::vector< double > &, const std::vector< double > &, const std::vector< double > &, const double &, const double &, const double &, const double &)

The documentation for this class was generated from the following files:

- Include/plotter.h
- Source/plotter.cpp

3.19 meshes::Point Struct Reference

Public Member Functions

- **Point** (double x_, double y_)
- bool **operator<** (const Point &other) const
- bool **operator==** (const Point &other) const
- bool **operator!=** (const Point &other) const
- double **dist** (const Point &other) const

Public Attributes

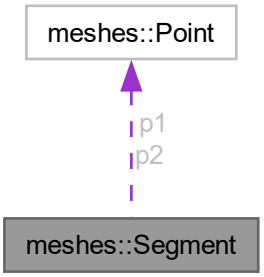
- double **x**
- double **y**
- double **epsilon** = 0.1

The documentation for this struct was generated from the following file:

- Include/mesh.h

3.20 meshes::Segment Struct Reference

Collaboration diagram for meshes::Segment:



Public Member Functions

- **Segment** (size_t ID_, Point p1_, Point p2_)
- bool **pointsApproximatelyEqual** (const Point &a, const Point &b) const
- bool **operator==** (const Segment &other) const
- bool **operator!=** (const Segment &other) const
- bool **shareCommonPoints** (const Segment &other) const
- std::vector< Point > **sharedPoints** (const Segment &other) const

Public Attributes

- Point **p1**
- Point **p2**
- int **ID**
- int **ID1** = -1
- int **ID2** = -1
- std::map< size_t, bool > **IDs**
- size_t **nNeighbors**
- double **epsilon** = 0.1001
- double **disk** = 0.1

The documentation for this struct was generated from the following file:

- Include/mesh.h

3.21 solver_params Struct Reference

Public Attributes

- unsigned **initial_guesses** = 12
- unsigned **midpoint_iterations** = 20
- bool **plot_inbetween** = false
- double **min_periastron** = 3.001
- bool **use_ellipse** = true

The documentation for this struct was generated from the following file:

- Include/utilities.h

3.22 Source Struct Reference

Public Attributes

- double **X**
- double **Y**
- double **impact_parameter**
- double **angle**
- double **z_factor**
- double **flux_o**

The documentation for this struct was generated from the following file:

- Include/BlackHole.h

3.23 Tests Class Reference

Static Public Member Functions

- static void **test_functions** (const double &, const double &)
- static void **test_BH_rendering** (const double &, const double &)
- static void **test_iso_radials** (const double &, const double &, const double &, const unsigned &)
- static void **test_iso_redshifts** (const double &, const double &, const double &, const int &, const unsigned &)

The documentation for this class was generated from the following files:

- Include/Tests.h
- Source/Tests.cpp

Chapter 4

File Documentation

4.1 BlackHole.h

```
00001 #pragma once
00002 #ifndef BLACKHOLE_H
00003 #define BLACKHOLE_H
00004 #include <iostream>
00005 #include <cmath>
00006 #include <fstream>
00007 #include <random>
00008 #include <algorithm>
00009 #include <unordered_map>
00010
00011 #include "TensorCalculus.h"
00012 #include "BlackHolePhysics.h"
00013 #include "IsoRadials.h"
00014 #include "IsoRedShift.h"
00015 #include "utilities.h"
00016
00017 struct Source {
00018     double X, Y, impact_parameter, angle, z_factor, flux_o;
00019 };
00020
00021 class BlackHole {
00022 public:
00023     // Constructor and other methods go here...
00024     BlackHole();
00025     BlackHole(double mass = 1.0, double inclination = 80, double acc = 1e-8);
00026     ~BlackHole();
00027     void sample_Sources(int n_points = 1000, const std::string& f = "", const std::string& f2 = "");
00028     void calc_isoradials(const std::vector<double>& direct_r, const std::vector<double>& ghost_r);
00029     void add_isoradial(Isoradial* isoradial, double radius, int order);
00030     std::map<double, IsoRedShift> calc_isoredshifts(std::vector<double> redshifts = { -0.15, 0.0, 0.1,
      0.2, 0.5 }, const int& order =0);
00031
00032 private://variables
00033     double inclination;
00034     double theta_0;
00035     double M;
00036     double rS;//Schwarzschild radius
00037     double Isco;// innermost stable circular orbit
00038     double acc;
00039     double disk_outer_edge;
00040     double disk_inner_edge;
00041
00042     //std::vector<double>  isoradials;
00043     //std::vector<double> isoredshifts;
00044     std::map<double, std::pair<int,Isoradial* > > isoradials;
00045     std::map<double, IsoRedShift> isoredshifts;
00046
00047     //make struct from this variables
00048     //std::unordered_map<std::string, double> settings;
00049     plot_params plot_params_;
00050     ir_params ir_parameters_;
00051     angular_properties angular_properties_;
00052     irs_solver_params irs_solver_params_;
00053     solver_params solver_params_;
00054
00055     /*int initial_guesses;
00056     int midpoint_iterations;
00057     bool plot_inbetween;
```

```
00058     bool use_ellipse;*/
00059     double min_periastron;
00060
00061     double critical_b;
00062     int angular_precision;
00063 private://methods
00064     Isoradial calc_apparent_outer_disk_edge();
00065     Isoradial calc_apparent_inner_disk_edge();
00066     double get_apparent_outer_edge_radius(Isoradial&, double angle, double rotation);
00067     double get_apparent_inner_edge_radius(Isoradial&, double angle, double rotation);
00068     std::pair<std::vector<double>, std::vector<double» apparent_inner_edge(Isoradial&, bool cartesian
      = true, double scale = 0.99);
00069     std::map<double, std::pair<int, Isoradial*»  get_dirty_isoradials(const int&);
00070 };
00071 #endif
```

4.2 BlackHolePhysics.h

```
00001 #pragma once
00002 #ifndef BLACKHOLEPHYSICS_H
00003 #define BLACKHOLEPHYSICS_H
00004 #include <iostream>
00005 #include <cmath>
00006 #include <vector>
00007 #include <cmath>
00008 #include <functional>
00009 #include <map>
00010 #include <boost/math/special_functions/jacobi_elliptic.hpp>
00011 #include "TensorCalculus.h"
00012 #include "utilities.h"
00013
00014 //using namespace std;
00015 //-------------------------------------------------------BLACK-HOLE-MATH.PY
00016
00017 class BHphysics
00018 {
00019 public:
00020     BHphysics();
00021     static double calc_q(double periastron, double bh_mass, double);
00022     static double calc_b_from_periastron(double periastron, double bh_mass, double);
00023
00024     static double k(double periastron, double bh_mass);
00025
00026     static double k2(double periastron, double bh_mass, double);
00027     static double zeta_inf(double periastron, double bh_mass, double tol);
00028     double zeta_r(double periastron, double r, double bh_mass);
00029     static double cos_gamma(double _a, double incl, double tol);
00030     void get_plot(const std::vector<double>&, const std::vector<double>&, const std::vector<double>&,
      const double&);
00031
00032     double cos_alpha(double phi, double incl);
00033
00034     double alpha(double phi, double incl);
00035     std::vector<double> filter_periastrons(const std::vector<double>& periastron, double bh_mass,
      double tol);
00036
00037     static double eq13(double periastron, double ir_radius, double ir_angle, double bh_mass, double
      incl, int n, double tol);
00038     static std::tuple<std::vector<double>, std::vector<double>, int> midpoint_method(
00039         const std::function<double(double, double, double, double, double, int, double)> func,
00040         const std::unordered_map<std::string, double>& args,
00041         const std::vector<double>& x,
00042         const std::vector<double>& y,
00043         int index_of_sign_change);
00044
00045     static double improve_solutions_midpoint(
00046         const std::function<double(double, double, double, double, double, int, double)>& func,
00047         const std::unordered_map<std::string, double>& args,
00048         const std::vector<double>& x,
00049         const std::vector<double>& y,
00050         int index_of_sign_change,
00051         int iterations
00052     );
00053
00054     static double calc_periastron(double _r, double incl, double _alpha, double bh_mass, int
      midpoint_iterations, bool plot_inbetween, int n, double min_periastron, int initial_guesses);
00055
00056     static double calc_impact_parameter(double _r, double incl, double _alpha, double bh_mass, int
      midpoint_iterations, bool plot_inbetween, int n, double min_periastron, int initial_guesses, bool
      use_ellipse);
00057
00058     double phi_inf(double periastron, double M);
00059
```

```
00060     double mu(double periastron, double bh_mass);
00061     static double ellipse(double r, double a, double incl);
00062 
00063     static double flux_intrinsic(double r, double acc, double bh_mass);
00064 
00065     static double flux_observed(double r, double acc, double bh_mass, double redshift_factor);
00066     static double redshift_factor(double radius, double angle, double incl, double bh_mass, double
      b_);
00067     static int find_index_sign_change_indices(const std::vector<double>&);
00068     //Black body temperature to RGB conversion
00069 
00070 static std::vector<double> wavelengthToRGB(const double& , const double& );//artist's impression
00072 //
00073 //  Tanner Helland formulas
00074 //
00075 static std::vector<double> convert_TH(const double& temperature, const double& brightness);
00076 
00077 
00078 
00080 //
00081 //  Neil Bartlett formulas
00082 //
00083 static std::vector<double> convert_NB(const double& temperature, const double& brightness);
00084 private:
00085 static  void normalizeRGB(std::vector<double>&, const double&);
00086 
00087 };
00088 #endif
```

4.3 isolines.h

```
00001 #pragma once
00002 #ifndef ISOLINES_H
00003 #define ISOLINES_H
00004 #include <algorithm>
00005 #include <cmath>
00006 #include <exception>
00007 #include <iostream>
00008 #include <limits>
00009 #include <memory>
00010 #include <utility>
00011 #include <vector>
00012 #include <map>
00013 #include <list>
00014 #include < iomanip >
00015 #include <unordered_map>
00016 #include <map>
00017 #include <stack>
00018 #include <queue>
00019 #include <deque>
00020 #include <chrono>
00021 
00022 #include <algorithm>
00023 #include <iostream>
00024 #include <limits>
00025 #include <set>
00026 #include <stdexcept>
00027 #include <vector>
00028 #include <dlib/threads.h>
00029 
00030 #include <memory>
00031 #include "mesh.h"
00032 using std::make_shared;
00033 
00034 
00035 using namespace dlib;
00036 using namespace meshes;
00037 // Define the maximum value for infinity
00038 //const double INF = std::numeric_limits<double>::max();
00039 
00040     class Isolines {
00041     public:
00042 
00043         Isolines(const std::shared_ptr<meshes::Mesh>&, const double&  );
00044 
00045         std::vector<Point>  get_iso_lines();
00046 
00047     public://variables
00048 
00049 
00050     private://methods
00051         std::vector<Segment> generateContourLines();
00052         bool isNoise(Segment);
```

```
00053         std::vector<std::vector<Point> > reduceToCurves(std::vector<Segment>&);
00054         void mergeSegments(std::vector<Segment>&);
00055         //void buildAdjacencyList(const std::vector<Segment>&);
00056         std::vector<Point> createPointsList(const std::vector<Segment>&);
00057         std::vector<Point> reorderPoints(const std::vector<Point>&);
00058         //std::pair<std::vector<double>, std::vector<double> > smoothCurve(const std::vector<Point>&);
00059         double distance(const Point&, const Point&);
00060         //std::vector<std::vector<Point> > constructCurveDFS(std::vector<Segment>& );
00061         std::vector<std::vector<double> > makeAdjacencyMatrix(const std::vector< Segment >&);
00062         std::vector<int> dijkstra(const std::vector<std::vector<double»&, int);
00063     private://variables
00064     const double contourValue;
00065     const std::shared_ptr<Mesh> meshPointer;
00066
00067     };
00068
00069 #endif
```

4.4 IsoRadials.h

```
00001 #pragma once
00002 #ifndef ISORADIALS_H
00003 #define ISORADIALS_H
00004 /*#include <iostream>
00005 #include <cmath>
00006 #include <fstream>
00007 #include <random>
00008 #include <algorithm>
00009 #include <unordered_map>*/
00010
00011 #include "TensorCalculus.h"
00012 #include "BlackHolePhysics.h"
00013 #include "utilities.h"
00014
00015 class Isoradial {
00016 public:
00017     Isoradial();
00018     Isoradial(double radius, double incl, double bh_mass, int order = 0);
00019     Isoradial(double radius, double incl, double bh_mass, int order, angular_properties);
00020     //Isoradial(const std::vector<double>& angles, const std::vector<double>& radius_b);
00021     std::pair<std::vector<double>, std::vector<double» get_bare_isoradials();
00022     void calculate_ISCO_region();
00023     std::pair<std::vector<double>, std::vector<double» get_ISCO_region();
00024     std::pair<std::vector<double>, std::vector<double» get_ISCO_curve();
00025     std::vector<double> get_redshift_factors();
00026     void calculate(const bool& isc=false);
00027     double get_radius();
00028
00029 private://variables
00030     double M;  // mass of the black hole containing this isoradial
00031     double rS;//Schwarzschild radius
00032     double rIsco;// innermost stable circular orbit
00033     double theta_0;  // inclination of the observer's plane
00034     double radius;
00035     int order;
00036     struct params {
00037         std::string param = "isoradial_solver_parameters";
00038     };
00039
00040     std::vector<double> redshift_factors;//TO DO: pack isoradials and redshift
00041     std::tuple<std::vector<double>, std::vector<double» cartesian_co;
00042     std::pair<std::vector<double>, std::vector<double> > bare_isoradials;//TEMPORARY for debugging:
      holds the polar coordinates (angles, radii) of the projected isoradial
00043     std::pair<std::vector<double>, std::vector<double> > ISCO_region;//TEMPORARY for debugging: holds
      the polar coordinates (angles, radii) of the projected isoradial
00044     std::pair<std::vector<double>, std::vector<double> > ISCO_boundary;
00045
00046 private://methods
00047
00048
00049 public://methods
00050
00051     std::pair<std::vector<double>, std::vector<double» calculate_coordinates(const bool& isc=false);
00052     std::vector<double> calc_redshift_factors();
00053     bool isPointInsidePolygon(const std::pair<double, double>&, const std::pair<std::vector<double>,
      std::vector<double> >&);
00054     std::vector<double> find_angle(double z);
00055     double get_b_from_angle(double angle);
00056     void calc_between(int ind);
00057     std::vector<double> force_intersection(double redshift);
00058     std::pair<std::vector<double>, std::vector<double» calc_redshift_location_on_ir(double redshift,
      bool cartesian = false);
00059
```

```
00060 public://variables ? make get method?
00061
00062     find_redshift_params find_redshift_params_;
00063     angular_properties angular_properties_;
00064     solver_params solver_params_;
00065     plot_params plot_params_;
00066     ir_params ir_params_;
00067
00068     std::vector<double> X;
00069     std::vector<double> Y;
00070     std::vector<double> _radii_b;
00071     std::vector<double> _angles;
00072 };
00073 #endif
```

4.5 IsoRedShift.h

```
00001 #pragma once
00002 #ifndef ISOREDSHIFTS_H
00003 #define ISOREDSHIFTS_H
00004 #include <iostream>
00005 #include <cmath>
00006 #include <fstream>
00007 #include <random>
00008 #include <algorithm>
00009 #include <unordered_map>
00010 #include <iostream>
00011 #include <cmath>
00012 #include <vector>
00013 #include <cmath>
00014 #include <functional>
00015 #include <map>
00016 #include <numeric>
00017
00018 #include <fstream>
00019 #include <iomanip>
00020
00021 #include "TensorCalculus.h"
00022 #include "BlackHolePhysics.h"
00023 #include "IsoRadials.h"
00024 #include "utilities.h"
00025 //#include "Delaunay.h"
00026 #include "mesh.h"
00027 #include "isolines.h"
00028 #include <dlib/threads.h>
00029 using namespace dlib;
00030 using namespace meshes;
00031
00032 #define xsect(p1,p2) (h[p2]*xh[p1]-h[p1]*xh[p2])/(h[p2]-h[p1])
00033 #define ysect(p1,p2) (h[p2]*yh[p1]-h[p1]*yh[p2])/(h[p2]-h[p1])
00034 //#define min(x,y) (x<y?x:y)
00035 //#define max(x,y) (x>y?x:y)
00036 struct cloud_points {
00037     cloud_points(double x, double y, double redshift) {
00038         this->x_ = x; this->y_ = y; this->redshift_ = redshift;
00039     }
00040     cloud_points() {}
00041     double redshift_;
00042     double x_;
00043     double y_;
00044 };/*
00045 struct Point {
00046     double x, y, rs;
00047     Point(double x_, double y_, double rs_) : x(x_), y(y_), rs(rs_) { ; }
00048 };
00049 struct Segment {
00050     Point p1;
00051     Point p2;
00052     size_t ID;//identification of Delaunay triangle
00053     Segment(size_t ID_, Point p1_, Point p2_) :
00054         ID(ID_), p1(p1_), p2(p2_)
00055     {
00056         ;
00057     }
00058 };*/
00059
00060 class IsoRedShift {
00061 public://methods
00062     IsoRedShift();
00063     IsoRedShift(const double& angle, const double& bh_mass, const double& lower_radius, const double&
      upper_radius, const size_t& _n_radii_, const size_t& _n_angles_, const double&);
00064     IsoRedShift(const double&, const double&, const double&, const std::map<double, std::pair<int,
      Isoradial*> >&);
```

```
00065     ~IsoRedShift();
00066     //std::multimap<double, std::vector<delaunay::Point> > get_isolines(const size_t n, const
      std::vector<double>&, const std::vector<double>&);
00067     std::multimap<double, std::vector<meshes::Point> > get_isolines(const size_t n);// , const
      std::vector<double>&, const std::vector<double>&));
00068     std::pair<std::vector<double>, std::vector<double» get_ISCO_curve();
00069     std::pair<std::vector<double>, std::vector<double» get_ConcaveHull();
00070     void improve();
00071 private://variables
00072     double theta_0;  // Inclination
00073     double redshift;
00074     double M;  // Black hole mass
00075     double rS;
00076     double rIsco;
00077     BHphysics BHp;
00078     std::vector<double> angles;
00079     std::vector<double> radii;
00080     size_t n_radii;
00081     size_t n_angles;
00082     double lower_radius;
00083     double upper_radius;
00084     std::vector<double> x;
00085     std::vector<double> y;
00086     double max_radius;
00087     irs_solver_params irs_solver_params_;
00088     find_redshift_params find_redshift_params_;
00089     angular_properties angular_properties_;
00090     solver_params solver_params_;
00091     plot_params plot_params_;
00092     ir_params ir_params_;
00093     std::pair<std::vector<double>, std::vector<double> > ISCO_boundary;
00094
00095 private://methods
00096     void make_grid();
00097     double calculateDistance(const meshes::Point& p1, const meshes::Point& p2);
00098     double findSmallestDistance(const std::vector<meshes::Point>&);
00099
00100 public://variables ? make get method?
00101     double redshift_;
00102     double x_;
00103     double y_;
00104     double x_max = -100000000000000.0;
00105     double x_min = -x_max;
00106     double y_max = x_max;
00107     double y_min = x_min;
00108     double redshift_max = -1000000000000.0;
00109     double redshift_min = -redshift_max;
00110     std::vector<double> xIsco;
00111     std::vector<double> yIsco;
00112     std::vector<meshes::Point> ConcaveHull;
00113     std::vector<meshes::Point> ISCO;
00114     double delta = 5.0e-2;
00115     double redshift_treshold;//TEMPORARY
00116     std::vector<double> xCoordinates;
00117     std::vector<double > yCoordinates;
00118     std::vector<double > redshifts;
00119     std::vector<double> xGrid;//TEMPORARY
00120     std::vector<double> yGrid;//TEMPORARY
00121     std::vector<std::vector<double» redshiftGrid;//TEMPORARY
00122 };
00123 #endif
```

4.6 mesh.h

```
00001 #pragma once
00002 #ifndef MESH_H
00003 #define MESH_H
00004 #include <algorithm>
00005 #include <cmath>
00006 #include <exception>
00007 #include <iostream>
00008 #include <limits>
00009 #include <memory>
00010 #include <utility>
00011 #include <vector>
00012 #include <map>
00013 #include <list>
00014 #include < iomanip >
00015 #include <unordered_map>
00016 #include <map>
00017 #include <stack>
00018 #include <queue>
00019 #include <deque>
```

```
00020 #include <chrono>
00021
00022 #include <algorithm>
00023 #include <iostream>
00024 #include <limits>
00025 #include <set>
00026 #include <stdexcept>
00027 #include <vector>
00028 #include <dlib/threads.h>
00029
00030 #include <memory>
00031
00032
00033 using namespace dlib;
00034 #define DIFFERENCE 0.0005
00035 #define EQ(_x_,_y_) (((_x_-_y_<DIFFERENCE)&&(_y_-_x_<DIFFERENCE))?1:0)
00036 // Define the maximum value for infinity
00037 const double INF = std::numeric_limits<double>::max();
00038
00039 namespace meshes {
00040     // Point structure representing a 2D Point
00041     struct Point {
00042         double x, y;
00043         double epsilon = 0.1; // Threshold distance for equality check
00044         Point(double x_, double y_) : x(x_), y(y_) {}
00045         // Custom comparison operators for set
00046         bool operator<(const Point& other) const {
00047             return x < other.x || (x == other.x && y < other.y);
00048         }
00049
00050         bool operator==(const Point& other) const {
00051             return std::abs(x - other.x) < epsilon && std::abs(y - other.y) < epsilon;
00052         }
00053         bool operator!=(const Point& other) const {
00054             return x != other.x || y != other.y;
00055         }
00056         double dist(const Point& other) const {
00057             return std::sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
00058         }
00059     };
00060
00061     struct driehoek {
00062         //driehoek() {};
00063         size_t ID;//identification -> not necessary for the moment being
00064         size_t ID1;//identification of neighbours Mesh triangle (driehoeken)
00065         size_t ID2;//identification of neighbours Mesh triangle (driehoeken)
00066         size_t ID3;//identification of neighbours Mesh triangle (driehoeken)
00067
00068         size_t nNeighbors;//how many has he got (could be two)
00069         Point p1;//1st triange Point
00070         Point p2;//2nd triange Point
00071         Point p3;//3rd triange Point
00072         Point Centroid;//centroid triange Point
00073
00074         double  Rs1;//redshift @ triange Point
00075         double  Rs2;//redshift @ triange Point
00076         double  Rs3;//redshift @ triange Point
00077
00078         driehoek(size_t ID_, Point p1_, Point p2_, Point p3_, double rs1_, double rs2_, double rs3_) :
00079             ID(ID_),
00080             p1(p1_),
00081             p2(p2_),
00082             p3(p3_),
00083             Centroid(p1_),
00084             Rs1(rs1_),
00085             Rs2(rs2_),
00086             Rs3(rs3_)
00087         {
00088             Centroid = Point((p1.x + p2.x + p3.x / 3), (p1.y + p2.y + p3.y / 3));
00089             ID1 = std::numeric_limits<size_t>::quiet_NaN();
00090             ID2 = std::numeric_limits<size_t>::quiet_NaN();
00091             ID3 = std::numeric_limits<size_t>::quiet_NaN();
00092         };
00093         // Function to check if two triangles share an edge
00094         bool share_edge(const driehoek& other) const {
00095             std::set<Point> shared_vertices{ p1, p2, p3 };
00096             return static_cast<int>(shared_vertices.count(other.p1) + shared_vertices.count(other.p2)
    + shared_vertices.count(other.p3) == 2);
00097         }
00098         // Function to check which edge two triangles share
00099         std::set<Point> shared_edge(const driehoek& other) const {
00100             std::set<Point> shared_vertices{ p1, p2, p3 };
00101             std::set<Point> common_vertices;
00102
00103             for (const Point& vertex : { other.p1, other.p2, other.p3 }) {
00104                 if (shared_vertices.count(vertex) > 0) {
00105                     common_vertices.insert(vertex);
```

```
00106                 }
00107             }
00108
00109             return common_vertices;
00110         }
00111     };
00112
00113     struct Segment {
00114         Point p1;
00115         Point p2;
00116         int ID;;//identification of Mesh triangle (driehoeken)
00117         int ID1 = -1;// = std::numeric_limits<size_t>::quiet_NaN();;//identification of neighbours
      Mesh triangle (driehoeken)
00118         int ID2 = -1;// = std::numeric_limits<size_t>::quiet_NaN();;//identification of neighbours
      Mesh triangle (driehoeken)
00119         std::map<size_t, bool> IDs;
00120         size_t nNeighbors;// = 0;//how many has he got (could be two)
00121         double epsilon = 0.1001; // Threshold distance for equality check
00122
00123         //bool check_equal(Point, Point);
00124         //Segment(); ;
00125         double disk = 0.1;
00126         Segment(size_t ID_, Point p1_, Point p2_) :
00127             ID(ID_), p1(p1_), p2(p2_)
00128         {
00129             ID1 = -1;
00130             ID2 = -1;
00131             nNeighbors = 0;
00132         }
00133         // Define a function to check if two points are approximately equal
00134         bool pointsApproximatelyEqual(const Point& a, const Point& b) const {
00135             double dx = a.x - b.x;
00136             double dy = a.y - b.y;
00137             return std::sqrt(dx * dx + dy * dy) < epsilon;
00138         }
00139         bool operator==(const Segment& other) const {
00140             return p1.x == other.p1.x && p1.y == other.p1.y && p2.x == other.p2.x && p2.y ==
      other.p2.y;
00141         }
00142         bool operator!=(const Segment& other) const {
00143             return p1.x != other.p1.x || p1.y != other.p1.y || p2.x != other.p2.x || p2.y !=
      other.p2.y;
00144         }
00145         // Function to check if two segments share any common points
00146         bool shareCommonPoints(const Segment& other) const {
00147             return (pointsApproximatelyEqual(p1, other.p1) || pointsApproximatelyEqual(p1, other.p2)
      || pointsApproximatelyEqual(p2, other.p1) || pointsApproximatelyEqual(p2, other.p2));
00148         }
00149         // Function to check if two segments share any common points
00150         std::vector<Point> sharedPoints(const Segment& other) const {
00151             std::vector<Point> shared;
00152             if (pointsApproximatelyEqual(p1, other.p1) || pointsApproximatelyEqual(p1, other.p2))
      shared.push_back(p1);
00153             if (pointsApproximatelyEqual(p2, other.p1) || pointsApproximatelyEqual(p2, other.p2))
      shared.push_back(p2);
00154             return shared;
00155         }
00156     };
00157
00158     //@see
      https://stackoverflow.com/questions/33333363/built-in-mod-vs-custom-mod-function-improve-the-performance-of-modulus-op/3
00159     inline size_t fast_mod(const size_t i, const size_t c) {
00160         return i >= c ? i % c : i;
00161     }
00162
00163     // Kahan and Babuska summation, Neumaier variant; accumulates less FP error
00164     inline double sum(const std::vector<double>& x) {
00165         double sum = x[0];
00166         double err = 0.0;
00167
00168         for (size_t i = 1; i < x.size(); i++) {
00169             const double k = x[i];
00170             const double m = sum + k;
00171             err += std::fabs(sum) >= std::fabs(k) ? sum - m + k : k - m + sum;
00172             sum = m;
00173         }
00174         return sum + err;
00175     }
00176
00177     inline double dist(
00178         const double ax,
00179         const double ay,
00180         const double bx,
00181         const double by) {
00182         const double dx = ax - bx;
00183         const double dy = ay - by;
00184         return dx * dx + dy * dy;
```

```
00185     }
00186
00187     inline double circumradius(
00188         const double ax,
00189         const double ay,
00190         const double bx,
00191         const double by,
00192         const double cx,
00193         const double cy) {
00194         const double dx = bx - ax;
00195         const double dy = by - ay;
00196         const double ex = cx - ax;
00197         const double ey = cy - ay;
00198
00199         const double bl = dx * dx + dy * dy;
00200         const double cl = ex * ex + ey * ey;
00201         const double d = dx * ey - dy * ex;
00202
00203         const double x = (ey * bl - dy * cl) * 0.5 / d;
00204         const double y = (dx * cl - ex * bl) * 0.5 / d;
00205
00206         if ((bl > 0.0 || bl < 0.0) && (cl > 0.0 || cl < 0.0) && (d > 0.0 || d < 0.0)) {
00207             return x * x + y * y;
00208         }
00209         else {
00210             return std::numeric_limits<double>::max();
00211         }
00212     }
00213
00214     inline bool orient(
00215         const double px,
00216         const double py,
00217         const double qx,
00218         const double qy,
00219         const double rx,
00220         const double ry) {
00221         return (qy - py) * (rx - qx) - (qx - px) * (ry - qy) < 0.0;
00222     }
00223
00224     inline std::pair<double, double> circumcenter(
00225         const double ax,
00226         const double ay,
00227         const double bx,
00228         const double by,
00229         const double cx,
00230         const double cy) {
00231         const double dx = bx - ax;
00232         const double dy = by - ay;
00233         const double ex = cx - ax;
00234         const double ey = cy - ay;
00235
00236         const double bl = dx * dx + dy * dy;
00237         const double cl = ex * ex + ey * ey;
00238         const double d = dx * ey - dy * ex;
00239
00240         const double x = ax + (ey * bl - dy * cl) * 0.5 / d;
00241         const double y = ay + (dx * cl - ex * bl) * 0.5 / d;
00242
00243         return std::make_pair(x, y);
00244     }
00245
00246     struct compare {
00247         std::vector<double> const& coords;
00248         double cx;
00249         double cy;
00250
00251         bool operator()(std::size_t i, std::size_t j) {
00252             const double d1 = dist(coords[2 * i], coords[2 * i + 1], cx, cy);
00253             const double d2 = dist(coords[2 * j], coords[2 * j + 1], cx, cy);
00254             const double diff1 = d1 - d2;
00255             const double diff2 = coords[2 * i] - coords[2 * j];
00256             const double diff3 = coords[2 * i + 1] - coords[2 * j + 1];
00257
00258             if (diff1 > 0.0 || diff1 < 0.0) {
00259                 return diff1 < 0;
00260             }
00261             else if (diff2 > 0.0 || diff2 < 0.0) {
00262                 return diff2 < 0;
00263             }
00264             else {
00265                 return diff3 < 0;
00266             }
00267         }
00268     };
00269
00270     inline bool in_circle(
00271         const double ax,
```

```
00272         const double ay,
00273         const double bx,
00274         const double by,
00275         const double cx,
00276         const double cy,
00277         const double px,
00278         const double py) {
00279         const double dx = ax - px;
00280         const double dy = ay - py;
00281         const double ex = bx - px;
00282         const double ey = by - py;
00283         const double fx = cx - px;
00284         const double fy = cy - py;
00285 
00286         const double ap = dx * dx + dy * dy;
00287         const double bp = ex * ex + ey * ey;
00288         const double cp = fx * fx + fy * fy;
00289 
00290         return (dx * (ey * cp - bp * fy) -
00291             dy * (ex * cp - bp * fx) +
00292             ap * (ex * fy - ey * fx)) < 0.0;
00293     }
00294 
00295     constexpr double EPSILON = std::numeric_limits<double>::epsilon();
00296     constexpr std::size_t INVALID_INDEX = std::numeric_limits<std::size_t>::max();
00297 
00298     inline bool check_pts_equal(double x1, double y1, double x2, double y2) {
00299         return std::fabs(x1 - x2) <= EPSILON &&
00300             std::fabs(y1 - y2) <= EPSILON;
00301     }
00302 
00303     // monotonically increases with real angle, but doesn't need expensive trigonometry
00304     inline double pseudo_angle(const double dx, const double dy) {
00305         const double p = dx / (std::abs(dx) + std::abs(dy));
00306         return (dy > 0.0 ? 3.0 - p : 1.0 + p) / 4.0; // [0..1)
00307     }
00308 
00309     struct MeshPoint {
00310         std::size_t i;
00311         double x;
00312         double y;
00313         std::size_t t;
00314         std::size_t prev;
00315         std::size_t next;
00316         bool removed;
00317     };
00318 
00319     class Mesh {
00320     public:
00321         //Mesh(void);
00322         //Mesh(std::vector<double> const&, std::vector<double> const&, std::vector<double> const&);
00323         Mesh(std::vector<double> const&, std::vector<double> const&, std::vector<double> const&, const
    std::vector<double>&, const std::vector<double>& yisco);
00324         //Mesh(const std::vector<double> , const std::vector<std::size_t> );
00325         Mesh(const Mesh& other);
00326         // Assignment operator
00327         Mesh& operator=(const Mesh& other) {
00328             if (this != &other) {
00329                 coords = other.coords;
00330                 triangles = other.triangles;
00331                 driehoeken = other.driehoeken;
00332                 x_coords = other.x_coords;
00333                 y_coords = other.y_coords;
00334                 redshift_field = other.redshift_field;
00335                 xISCO = other.xISCO;
00336                 yISCO = other.yISCO;
00337                 xIscoMax = other.xIscoMax;
00338                 xIscoMin = other.xIscoMin;
00339                 yIscoMax = other.yIscoMax;
00340                 yIscoMin = other.yIscoMin;
00341                 ISCO = other.ISCO;
00342                 IscoEpsilon = other.IscoEpsilon;
00343                 concaveHull = other.concaveHull;
00344                 nConcaveHull = other.nConcaveHull;
00345             }
00346             return *this;
00347         }
00348 
00349         std::vector<std::size_t> getMesh();
00350         std::vector<double> getCoords();
00351         void triangulate();
00352         double get_hull_area();
00353         std::vector<double> get_hull_coords();
00354         std::vector<size_t> get_hull_points();
00355 
00356         double edge_length(size_t e);
00357         size_t get_interior_point(size_t e);
```

```
00358         std::vector<double> concavehull(double chi_factor = 0.1);
00359         std::vector<std::size_t> get_triangles();
00360         std::vector<double> get_tri_coordinates();
00361 
00362     public://variables
00363         std::vector<double> coords;
00364         std::vector<std::size_t> triangles;
00365         std::vector<std::size_t> halfedges;
00366         std::vector<std::size_t> hull_prev;
00367         std::vector<std::size_t> hull_next;
00368         std::vector<std::size_t> hull_tri;
00369         std::size_t hull_start;
00370     public://methods
00371         std::size_t legalize(std::size_t a);
00372         std::size_t hash_key(double x, double y) const;
00373         std::size_t add_triangle(
00374             std::size_t i0,
00375             std::size_t i1,
00376             std::size_t i2,
00377             std::size_t a,
00378             std::size_t b,
00379             std::size_t c);
00380         void link(std::size_t a, std::size_t b);
00381         size_t  next_halfedge(size_t e);
00382 
00383         size_t prev_halfedge(size_t e);
00384         std::pair<Segment, Segment> findConvexHullSegments(const std::vector<double>&, const
      std::vector<double>&);
00385 
00386         bool isOnPolygon(const Point&, const Point&, const Point&);
00387         bool oneIsOnPolygon(const Point&);
00388         bool is_left(const Point& a, const Point& b, const Point& c);
00389         bool isOutsideConcaveHull(const Point&);
00390         //void makeISCO(const std::vector<double>& , const std::vector<double>& );
00391         void makeISCO();
00392         void make_driehoeken();
00393 
00394     public://variables
00395         std::vector<std::size_t> m_hash;
00396         double m_center_x;
00397         double m_center_y;
00398         std::size_t m_hash_size;
00399         std::vector<std::size_t> m_edge_stack;
00400         std::vector<double>  x_coords;
00401         std::vector<double>  y_coords;
00402         std::vector<double> redshift_field;
00403         std::vector<driehoek> driehoeken;
00404         std::vector<Point> concaveHull;
00405         int nConcaveHull;
00406         std::vector<double>  xISCO;
00407         std::vector<double>  yISCO;
00408         double xIscoMax;// = *std::max_element(xisco.begin(), xisco.end());
00409         double xIscoMin;// = *std::min_element(xisco.begin(), xisco.end());
00410         double yIscoMax;// = *std::max_element(xisco.begin(), xisco.end());
00411         double yIscoMin;// = *std::min_element(xisco.begin(), xisco.end());
00412         double IscoEpsilon=0.0;// = 0;
00413         std::vector<Point> ISCO;
00414     };
00415 
00416 } //namespace meshes
00417 #endif
```

4.7 plotter.h

```
00001 #pragma once
00002 #ifndef PLOTTER_H
00003 #define PLOTTER_H
00004 //#include "winuser.h"
00005 #include <vector>
00006 #include <iostream>
00007 #include <fstream>
00008 #include <string>
00009 #include <type_traits>
00010 #include <sstream>
00011 #include <array>
00012 #include <exception>
00013 #include <stdexcept>
00014 #include <utility>
00015 #include <unordered_map>
00016 #include <algorithm>
00017 #include <map>
00018 
00019 #include "IsoRedShift.h"
```

```
00020
00021
00022
00023
00024 #include <discpp.h>
00025 #include <Windows.h>
00026 // Undefine macros that may cause conflicts
00027 #undef min
00028 #undef max
00029 const double a_PI = 3.14159265358979323846;
00030
00031 class Plotter {
00032 public:
00033     Plotter();
00034     ~Plotter();
00035     void plot_isoradials(std::vector<double>&, std::vector<double>&, std::vector<double>&,
      std::vector<double>&);//bare isoradials
00036     void plot_isoradials(double, std::vector<double>&, std::vector<double>&, std::vector<double>&,
      std::vector<double>&, std::vector<double>&, std::vector<double>&,bool =false);//isoradials with
      redshift
00037     void plot_iso_redshifts(const double&, const std::multimap<double, std::vector<meshes::Point> >&,
      const double&, const double&, const double&, const double&, const std::pair<std::vector<double>,
      std::vector<double>&, const std::pair<std::vector<double>, std::vector<double>&,const bool&);
00038     void plot_iso_redshifts(const double&, const std::vector<double>&, const std::vector<double>&,
      const std::vector<double>&, const double&, const double&, const double&, const double&);//
00039     //void plot_iso_redshifts(IsoRedShift& Irs, const std::vector<std::vector<double» irsgrid, const
      double& inclination, std::multimap<double, std::vector<std::pair<std::vector<double>,
      std::vector<double> > > >& isolines, const double& x_max_, const double& x_min_, const double& max_rs,
      const double& min_rs, const bool& loop);
00040     //void plot_iso_redshifts(const double&, const std::vector<std::vector<double»&, const
      std::vector<double>& X, const std::vector<double>& Y, const double&, const double&, const double&,
      const double&, const bool&);
00041     //void plot_iso_redshifts(const double&, const std::vector<double>&, const std::vector<double>& ,
      const std::vector<double>& , const double&, const double&, const double&, const double&, const bool&);
00042
00043 private:// Functions to convert a value to RGB format
00044     std::vector<std::tuple<double, double, double> >convertToRGB(std::vector<double>);
00045     std::vector<std::tuple<double, double, double> >convertToRGBbis(const std::vector<double>&, const
      double&);
00046     std::vector<double> normalize_vector(std::vector<double>);
00047     std::vector<double> flatten_matrix(const std::vector<std::vector<double> >&);
00048 private://variables
00049     int screenWidth;
00050     int screenHeight;
00051     Dislin* g;
00052     int ic;//dislin
00053     double x_max;
00054     double x_min;
00055     double y_max;
00056     double y_min;
00057     int Npoints;
00058     // Vectors to store RGB color values
00059     std::vector<std::tuple<double, double, double» rgbVector;
00060     std::vector<std::tuple<double, double, double» rgbVector_g;
00061
00062 };
00063 #endif
```

4.8 TensorCalculus.h

```
00001 #pragma once
00002 #ifndef TENSORCALCULUS_H
00003 #define TENSORCALCULUS_H
00004
00005 /*
00006 General library in progress with common operators
00007 To extend with tensor algebra operations.
00008 */
00009 #include <iostream>
00010 #include <vector>
00011 #include <tuple>
00012 #include <functional>
00013 #include <cmath>
00014
00015 #include "utilities.h"
00016 //#include <Eigen/Dense>
00017
00018 //using namespace Eigen;
00019
00020 //const double M_PI = 3.14159265358979323846;
00021 class OperatorsOrder2 {
00022 public:
00023     OperatorsOrder2(int nGrid, double delta);
```

```
00024     ~OperatorsOrder2();
00025
00026     std::vector<std::vector<std::vector<double>> laplace(const
      std::vector<std::vector<std::vector<double>>& fct);
00027     std::tuple<std::vector<std::vector<std::vector<double>>,
      std::vector<std::vector<std::vector<double>>>, std::vector<std::vector<std::vector<double>>>
      gradient(const std::vector<std::vector<std::vector<double>>& fct);
00028     std::vector<std::vector<std::vector<double>> divergence(const
      std::vector<std::vector<std::vector<double>>& v_x,
00029         const std::vector<std::vector<std::vector<double>>& v_y,
00030         const std::vector<std::vector<std::vector<double>>& v_z);
00031     std::tuple<std::vector<std::vector<std::vector<double>>,
      std::vector<std::vector<std::vector<double>>>, std::vector<std::vector<std::vector<double>>>
      gradDiv(const std::vector<std::vector<std::vector<double>>& A_x,
00032         const std::vector<std::vector<std::vector<double>>& A_y,
00033         const std::vector<std::vector<std::vector<double>>& A_z);
00034     std::tuple<double, double, double> partialDerivs(const
      std::vector<std::vector<std::vector<double>>& fct, int i, int j, int k);
00035     std::vector<std::vector<std::vector<double>> Add(const
      std::vector<std::vector<std::vector<double>>&, const double&, const
      std::vector<std::vector<std::vector<double>>&, const double&);
00036     static std::vector<double>  linspace(const double&, const double&, const int&);
00037     static std::vector<double> nonLinSpace(const bool&, const double&, const int&);
00038     static double phi(const double&, const double&);
00039     static std::vector<double>  logspace(const double&, const double&, const int&);
00040     //static std::vector<double> ellipticspace(const bool&, const double& , const int& );
00041
00042     static std::pair<std::vector<double>, std::vector<double>> polar_to_cartesian_lists(const
      std::vector<double>& radii, const std::vector<double>& angles, const double& rotation);
00043     static std::pair<double, double> polar_to_cartesian_single(double th, double radius, double
      rotation);
00044     //static std::vector<double> polar_to_cartesian_single_as_vector(double th, double radius, double
      rotation);
00045     static std::pair<double, double> cartesian_to_polar(double x, double y);
00046     static double get_angle_around(const std::vector<double>& p1, const std::vector<double>& p2);
00047
00048 private:
00049     static std::vector<double> matrixMultiply(const std::vector<std::vector<double>>& matrix, const
      std::vector<double>& vector);
00050     int nGrid;
00051     double delta;
00052 };
00053 #endif
```

4.9 Tests.h

```
00001 #pragma once
00002 #ifndef TESTS_H
00003 #define TESTS_H
00004
00005 #include <vector>
00006 #include <iostream>
00007 #include <fstream>
00008 #include <string>
00009 #include <type_traits>
00010 #include <sstream>
00011 #include <array>
00012 #include <exception>
00013 #include <stdexcept>
00014 #include <utility>
00015 #include <unordered_map>
00016 #include <algorithm>
00017 #include "BlackHole.h"
00018 #include "BlackHolePhysics.h"
00019 #include "plotter.h"
00020 #include <conio.h>
00021 #include <iostream>
00022 #include <memory>
00023 #include <chrono>
00024
00025 //const double M_PI = 3.14159265358979323846;
00026 class Tests{
00027     public:
00028 static void test_functions(const double&, const double&);
00029 static void test_BH_rendering(const double&, const double&);
00030 static void test_iso_radials(const double&, const double&, const double&, const unsigned&);
00031 static void test_iso_redshifts(const double&, const double&, const double&, const int &,const
      unsigned&);
00032 };
00033 #endif
```

4.10 utilities.h

```
00001 #pragma once
00002 #ifndef UTILITIES_H
00003 #define UTILITIES_H
00004
00005 #include <vector>
00006 #include <iostream>
00007 #include <fstream>
00008 #include <string>
00009 #include <type_traits>
00010 #include <sstream>
00011 #include <array>
00012 #include <exception>
00013 #include <stdexcept>
00014 #include <utility>
00015 #include <unordered_map>
00016 #include <algorithm>
00017
00018 const double M_PI = 3.14159265358979323846;
00019 //INI settings are wrapped in struct
00020 struct irs_solver_params {
00021     unsigned initial_guesses = 12;
00022     unsigned midpoint_iterations = 12;
00023     double  times_inbetween = 2;// amount of times to double the precision of an isoredshift line when
      improving
00024     unsigned retry_angular_precision = 15;//angular precision to calculate isoradials with when
      improving solutions
00025     double  min_periastron = 3.01;// minimum distance to black hole(must be strictly larger than 3M),
      in units of black hole mass(photon sphere is at 3M)
00026     bool use_ellipse = true;
00027     unsigned retry_tip = 50;
00028     unsigned initial_radial_precision = 15;
00029     bool plot_inbetween = false;// plot isoredshifts while improving them
00030     double angular_margin = 0.3;
00031 };
00032
00033 struct angular_properties {
00034     double start_angle = 0.0;
00035     double end_angle = M_PI;
00036     unsigned angular_precision =  500;
00037     bool mirror =  true;
00038 };
00039
00040 struct ir_params {
00041     double start_angle = 0.0;
00042     double  end_angle = M_PI;
00043     unsigned angular_precision =  500;
00044     bool    mirror =  true;// if True, calculates only half of the isoradial and mirrors it
00045     double angular_margin = 0.3;
00046 };
00047
00048 struct plot_params {
00049     bool plot_isoredshifts_inbetween = false;
00050     bool save_plot = false;
00051     bool plot_ellipse = false;
00052     bool plot_core = true;
00053     bool redshift = true;
00054     std::string linestyle = "-";
00055     double  linewidth = 1.;
00056     std::string key = "";
00057     std::string face_color = "black";
00058     std::string line_color = "white";
00059     std::string text_color = "white";
00060     double alpha = 1.;
00061     bool show_grid = false;
00062     bool legend = false;
00063     bool orig_background = false;
00064     bool plot_disk_edges = false;
00065     std::pair<double, double>   ax_lim = { -100, 100 };
00066     std::string title = "Isoradials for R =";
00067 };
00068
00069 struct solver_params {
00070     unsigned initial_guesses = 12;
00071     unsigned midpoint_iterations = 20;
00072     bool plot_inbetween = false;// plot isoredshifts while improving them
00073     double min_periastron = 3.001;// minimum distance to black hole, in units of black hole
      mass(photon sphere is at 3M)
00074     bool use_ellipse = true;
00075 };
00076
00077 struct  find_redshift_params {
00078     bool force_redshift_solution = false;
00079     unsigned max_force_iter = 5;
00080 };
00081
```

```
00082 class CSVRow
00083 {
00084 public:
00085     std::string_view operator[](std::size_t index) const
00086     {
00087         return std::string_view(&m_line[m_data[index] + 1], m_data[index + 1] - (m_data[index] + 1));
00088     }
00089     std::size_t size() const
00090     {
00091         return m_data.size() - 1;
00092     }
00093     void readNextRow(const std::string& line)
00094         //void readNextRow(std::istream& str)
00095     {
00096         //std::getline(str, m_line);
00097         m_line = line;
00098         //std::cout « m_line « std::endl;
00099         m_data.clear();
00100         m_data.emplace_back(-1);
00101         std::string::size_type pos = 0;
00102         while ((pos = m_line.find(',', pos)) != std::string::npos)
00103         {
00104             m_data.emplace_back(pos);
00105             ++pos;
00106         }
00107         // This checks for a trailing comma with no data after it.
00108         pos = m_line.size();
00109         m_data.emplace_back(pos);
00110     }
00111 private:
00112     std::string         m_line;
00113     std::vector<int>    m_data;
00114 };
00115 /*
00116 std::istream& operator»(std::istream& str, CSVRow& data)
00117 {
00118     data.readNextRow(str);
00119     return str;
00120 }*/
00121
00122 #endif
```