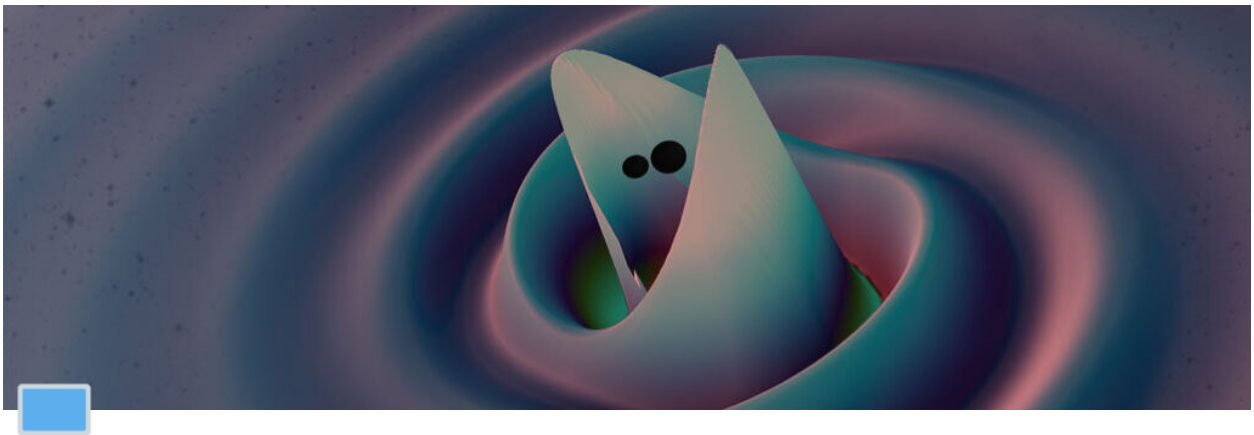


workspace

https://open.spotify.com/embed/playlist/6OrqVnO3qLwcl84L7VZiM8

▼ Helpful Commands

C++

January 19, 2021 — Individually

▼ TOV solution Task 1

1. I made a function that outputs different radius values and plotted the results on jupyter-notebook

```
/home/jennifer/TOVsolution/tasks/task1
```

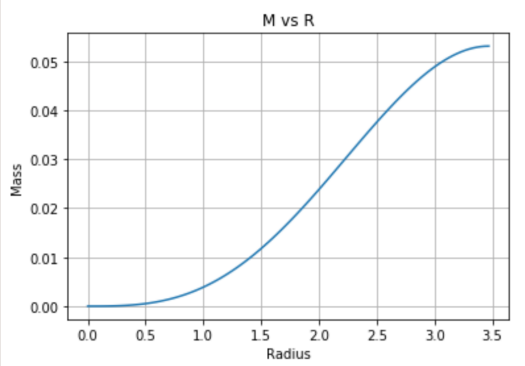
January 20, 2021 — Meeting with Teresa

▼ TOV solution Task 1:

1. We tried to write a function that allows the output of the Tov.cpp to be written in a file.
2. Ended meeting early because Teresa had some internet connection problems (she couldn't even write on her terminal). Rejoined later, and I got it to work! This is the code

```
  // This will integrate from r=0 to r=outer_radius.
  const double dr = outer_radius_/100.0; //current radius (the radius will be changing every iteration).
  //100.0 means how many steps
  double r = 0.0;
  std::ofstream ofs1 ("radius.txt", std::ofstream::out);
  std::ofstream ofs2 ("mass_of_r_over_r.txt", std::ofstream::out);
  std::ofstream ofs3 ("mass_of_r.txt", std::ofstream::out);
  while (r < outer_radius_) {
    ofs1 << r <<","; //tells us the mass inside the given radius over the radius: m(r)/r
    ofs2 << mass_over_radius(r) << ",";
    ofs3 << mass_over_radius(r) * r << ",";
    r += dr;
    }
  ofs1.close();
  ofs2.close();
  ofs3.close();
}
```

3. I plotted the results and this the plot:



4. To-dos:

- [x] ~~func that allows output of Tov.cpp to be written in a file~~
- [x] ~~plot const double dr = outer_radius_/50.0;~~

▼ **January 21, 2021; Thursday Lovelace Meeting**

- Tasks:
- Make a list of tasks on a slide from meeting with Nils and Lovelace
- Include the harmonic oscillator on the Tov.cpp file with the radii (pretend that's time or something) then make sure we ge the right solution
-

February 2, 2021—Individually

▼ TOV solution Task 2:

Task 2: Include the boost_ode.cpp into the Tov.cpp file and make sure it prints correct results.

```
/home/jennifer/TOVsolution/tasks/task2
```

- now edit Tov.cpp to include radius

February 9, 2021— Meeting with Teresa

▼ TOV Solution Task 2:

- In the Tov test (test_Tov.cpp), you can comment everything and contruct the tov solution class with a specific central mass density, eos, etc.
- get the ODE for H. solve for

May 5, 2021— Meeting with Alex

▼ TOV solution Task 4:

- Error message:

```
/home/jennifer/TOVSolution/src/PointwiseFunctions/AnalyticSolutions/GeneralRelativity/Tov.cpp:241:16: error: 'this' cannot be implicitly
aptured in this context
           -2*mass_over_radius* current_position[1] - 4.0 * current_position[0];
              ^
/home/jennifer/TOVSolution/src/PointwiseFunctions/AnalyticSolutions/GeneralRelativity/Tov.cpp:241:16: error: reference to non-static mem
r function must be called
           -2*mass_over_radius* current_position[1] - 4.0 * current_position[0];
              ^~~~~~~~~~~~~~~~
2 errors generated.
```

error message when trying to include of the functions in gr::Solutions::TovSolution

- Cause of error message? Two reasons:

1. **We did not give the function an appropriate argument.**

we originally had this on L241

```
-2*mass_over_radius* current_position[1] - 4.0 * current_position[0];
```

but according to the member function documentation:

double gr::Solutions::TovSolution::mass_over_radius(double r)const

The mass inside the given radius over the radius $\frac{m(r)}{r}$.

Note

`r` should be non-negative and not greater than `outer_radius()`.

`mass_over_radius` requires arguments inside—specifically a double. so our new L241 should read something like:

```
-2*mass_over_radius(4.0)* current_position[1] - 4.0 * current_position[0];
```

where we have given a double or

```
-2*mass_over_radius(current_radius_r)* current_position[1] - 4.0 * current_position[0];
```

where `current_radius_r` was defined previously in the code as a double.

2. The lambda function required `this`

okay this one is harder to explain because I don't completely understand it myself, but let's start off with what is `this`?

- When inside a non-`static` function definition of a class, `this` is a pointer to the current object of that class and can be used explicitly (written out) or implicitly (used even if you don't type it out)
- When inside a non-`static` function definition of a class, `this->` is implicitly used when accessing a member variable of the class or when calling a non-`static` function (see example code snippet below).
- e.g. when calling `mass_over_radius(current_radius_r)` from within the `TovSolution` constructor, `this->mass_over_radius(current_radius)` is being implicitly called, which means the `mass_over_radius` function is being called on the current `TovSolution` object being constructed by the constructor
- Because the lambda being defined from lines 230-249 calls a non-`static` `TovSolution` class function (i.e. `mass_over_radius` at line 241) from within a non-`static` context in the class (i.e. line 241 is inside a `TovSolution` constructor definition, and constructors are non-`static`), `this->` is implicitly used when calling `mass_over_radius`, which means the lambda needs access to `this`, which means `this` needs to be captured by the lambda, which means `this` needs to be added to the `[]` part of the lambda on line 230.

TLDR: because the lambda is inside a non-`static` `TovSolution` class function (the constructor) and it calls a non-`static` `TovSolution` class function (`mass_over_radius`), the lambda needs to capture `this` to be able to call that function (`mass_over_radius`)Note: `this` does not exist inside a `static` member function definition in a class because when a class member function is `static`, that means it belongs to the class, not any one object from the class. They are not called with objects of the class, so there is no notion of "the current object" (`this`) when inside a `static` member function definition in a class

- Example code regarding this being used implicitly when referring to class member variables and when calling non-static class functions from within a non-static function of the same class:

```
#include <iostream>
// On ocean, save this file as something like this.cpp and compile it with:
//   clang++ this.cpp -o this
// Then run with:
//   ./this
class Rectangle {
  public:
    // Rectangle class constructor (constructors are non-static)
    Rectangle(double l, double w) {
        // If member variables of a class, like length and width here, are referred
        // to from within a non-static context, like a constructor (here) or a
        // non-static function, then 'this->' is implicitly used when using those
        // member variables, even if you don't explicitly type it out:
        length = l; // implicitly, this line is doing: this->length = l;
        width = w;  // implicitly, this line is doing: this->width = w;
```

```
        // More generally:
        //
        // Here inside a constructor, 'this' refers to the Rectangle object being
        // constructed when this constructor is called. e.g. When it's called in the first
        // line in main, any reference to 'this' inside this constructor will refer to
        // that Rectangle, my_rectangle, being constructed.
    }
    // non-static Rectangle class function that computes its area
    double get_area() {
        // See comment in Rectangle constructor above
        return length * width; // implicitly: return this->length * this->width;
    }
    // non-static Rectangle class function that prints the area by calling the
    // non-static Rectangle class function, get_area
    void print_area() {
        // If the non-static function, get_area, is called from within a non-static
        // of the same class like we are here, in print_area, 'this->' is implicitly
        // used with calling the function, even if you don't explicitly type it out.
        //
        // i.e. the following line commented out is equivalent to the one below it:
        // std::cout << "The area of the rectangle is: " << this->get_area() << std::endl;
        std::cout << "The area of the rectangle is: " << get_area() << std::endl;
        // More generally:
        //
        // Here inside a non-static function, 'this' refers to the Rectangle that this
        // print_area function is called on. e.g. When it's called in the second
        // line in main, any refernce to 'this' inside this function will refer to
        // that Rectangle, my_rectangle.
    }
    // class member variables
    double length;
    double width;
};
int main() {
    Rectangle my_rectangle(5.0, 10.0);  // call Rectangle constructor to construct Rectangle object
    my_rectangle.print_area();  // call Rectangle instance function (non-static function) that prints its area
}
```