# Horizon Finder - Summary

Yari De Paep

**Supervisors**: Frederik De Ceuster, Arthur Offermans, Tjonnie Li

*Note from the author – Legend explanation:*
*Shown in **red** are the names of the notebooks that have been used for that particular section.*
*Text in **blue** indicates tasks that have not been implemented or tried yet.*

## 1 Introduction

This report gives a summary of the work that has been done for a summer project on a horizon finder solver. The main purpose of this summary is to guide the reader through the different developments of this project, and to provide the reader with the necessary information to continue working on this project. This summary goes together with different Jupyter notebooks containing Python scripts on all components and details about the solving procedure.

## 2 First attempts

I started out quite ambitiously with the attempt at recreating the method in the paper [1]. This is a multigrid method, which requires several components to be solved (V-cycle, smoother, solver, ...). We came to the conclusion that this would probably be too difficult to implement all by myself from scratch, and that it would also take too much time. For that reason, we decided to take a different and simpler approach, namely the PyTorch library, which can solve the differential equation more explicitly. Here I also had an example available of a simpler scenario, such that I could base the code on this example. First, I decided to make a script/notebook which calculates the expansion function symbolically.

## 3 Symbolic calculation

First I developed a script which calculates the expansion function symbolically from equation (2) in the paper

$$\Theta \equiv (D_i s^i + K_{ij} s^i s^j - K)/\sqrt{2}, \tag{1}$$

where $D_i$ is the covariant derivative corresponding to the flat metric $\gamma_{ij}$ and $K_{ij}$ is the extrinsic curvature, and in which

$$s_i = \lambda(1, -\partial_\theta h, -\partial_\phi h), \tag{2}$$

with $h(\theta, \phi)$ the horizon function. Details about these definitions can be found in the paper.

The script exists in two versions depending on the available input. The two situations are:

1. Input $= \alpha, \beta_i, \gamma_{ij} \rightarrow$ AppHor_eq.ipynb
2. Input $= \alpha, \beta_i, \psi \rightarrow$ AppHor_conformal.ipynb
   This notebook contains all necessary steps and explanations for the calculation.

In both cases $\alpha$ and $\beta_i$ refer to the 3+1 ADM lapse function and shift vector respectively. $\gamma_{ij}$ is the spatial part of the metric and $\psi$ is the conformal factor, assuming a conformally flat metric. Furthermore a maximal slicing condition is assumed such that $K = 0$. In this work the Schwarzschild metric in isotropic coordinates has been used as a test case, so the second case notebook is used for calculating $\Theta$.

# 4 PyTorch solver

As mentioned before, I chose an approach using the PyTorch library because I already had an example available. In this way, we would be able to quickly obtain a first set of solutions without too much effort. Disclaimer: this method has only been worked out for three cases:

1. Full spherical symmetry $\rightarrow h(\theta, \phi) = h = cte$
2. Axisymmetric in $\theta \rightarrow h(\theta, \phi) = h(\phi)$
3. Axisymmetric in $\phi \rightarrow h(\theta, \phi) = h(\theta)$

## 4.1 Methodology

The solving procedure can be described by a multi-step process. All steps can be found in the notebook

AppHor_torch.ipynb

with short explanations. It is important to know that this script only works for the cases with symmetry in either the $\theta$- or $\phi$-coordinate. The main steps are the following:

1. Calculate the expansion function symbolically (see section 3)
2. The expression is converted to a string such that all operators (like derivatives, sqrts, ...) can be replaced with their PyTorch counterpart
3. Calculate loss function (residual to be minimized by steepest gradient descent)
    - Residual is the mean-squared of the whole expression evaluated on h
4. Solve for h iteratively (details in notebook in "solve_for_h"-function)

## 4.2 Results

### 4.2.1 Case 1: symmetry in $\theta$

Because of symmetry, we work in the plane $\theta = \frac{\pi}{2}$. The results below use a grid of $N = 50$ points in $\phi$ and an initial condition $h(\phi_i) = 0.9$ for every $i$ (the exact solution for $h$ is of course the Schwarzschild radius, in this case $0.5M$, and we took $M = 2$). The derivatives are approximated by forward finite differences (which is built-in in PyTorch), and we apply periodic boundary conditions because the points $\phi = 0$ and $\phi = 2\pi$ can be identified.
As long as the initial condition for $h$ is close enough to the actual solution, the method converges. Results for the above conditions can be seen on the figure below.

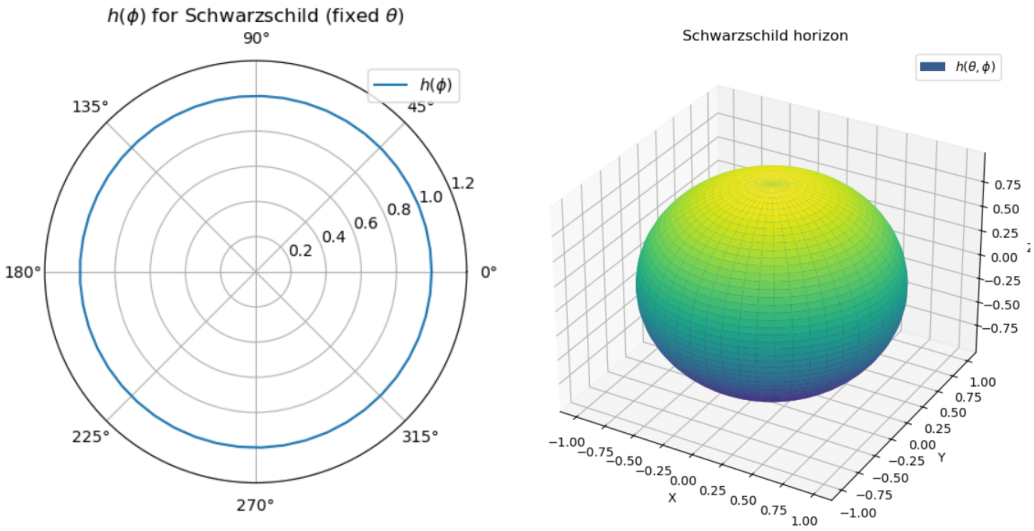


Figure 1: Left: 1D solution for $h(\phi)$ on the domain of $\phi$. Right: generalized solution to 3D (using symmetry).

### 4.2.2 Case 2: symmetry in $\phi$

The procedure works exactly the same in this case, with $\theta$ only ranging from 0 to $\pi$. Another important difference is that we can no longer apply periodic boundary conditions. Instead, we use the assumption that the solution at $\phi + \pi$ should be the same. The result is shown below.

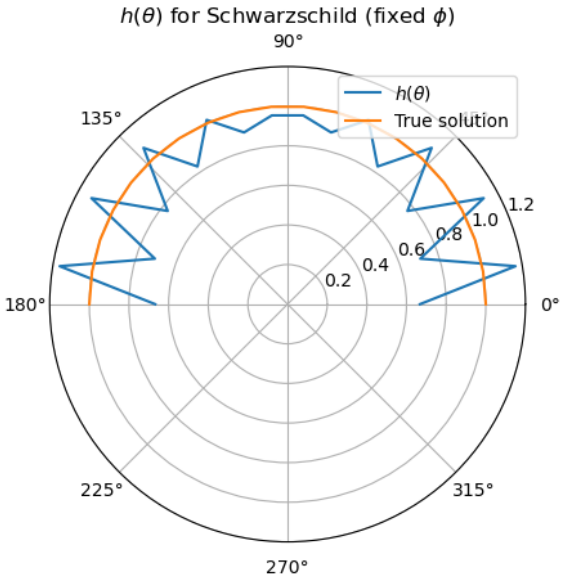


Figure 2: Solution of $h(\theta)$ found by PyTorch (blue line), and the analytical solution (orange line).

From this figure it is clear that the solver is not able to find the correct solution. A reoccurring problem is that the solution shows a saw-pattern each time, instead of being smooth.

We have not been able to figure out what exactly is the cause of this behavior.

Outlook:
We could still try to impose as boundary conditions a zero-derivative on the edge points 0 and $\pi$, such that there can be no discontinuity at the poles.

### 4.3 Spectral decomposition

Here we chose to decompose $h(\theta)$ into a linear combination of smooth basis functions in order to try to get rid of the discontinuous saw-pattern in the $\theta$-solution above. We decomposed $h(\theta)$ in a Fourier basis as

$$h(\theta) = A + \sum_{k=1}^{N} B_k \sin(k\theta) + \sum_{l=1}^{N} C_l \cos(l\theta). \tag{3}$$

This ansantz is then substituted into the expansion equation, and then solved to $A, B_k$ and $C_l$ using the same steepest gradient descent method as before. Details can be found in the notebook

AppHor_torch-spectral.ipynb

The solution in this case has a hard time finding a constant solution. Possible causes can be the choice of basis functions.

Outlook:
Decompose $h(\theta)$ in a different basis

# 5 Linear solver

In this section we step away from the PyTorch solver, and we go for a plan B instead inspired on the paper (cite). In here the authors show that it is possible to separate from the expansion function a spherical Laplacian (+ linear term) defined as

$$\Delta_{\theta\phi} h - 2h = \frac{\partial^2 h}{\partial \theta^2} + \cot\theta \frac{\partial h}{\partial \theta} + \frac{1}{\sin^2\theta} \frac{\partial^2 h}{\partial \phi^2} - 2h. \tag{4}$$

Since this expression is linear in $h$, its finite difference approximation can be represented as a matrix, say $A$ (in general a sparse matrix). Solving the expansion equation is then equivalent to solving

$$\Theta(h) = 0 \Leftrightarrow A \cdot h = S(h), \tag{5}$$

where $S(h)$ contains all remaining terms (most/all of which non-linear in $h$). This system can then explicitly be solved iteratively using the following steps:

1. Choose an initial guess $h_0$ and calculate $S(h_0)$
2. Use this result to update $h$ via $h_{new} = A^{-1} \cdot S(h_0)$
3. Now use the updated $h_{new}$ to calculate $S(h_{new})$, and the process can repeat itself up to a desired accuracy

We start again by considering the simpler cases with symmetry in one of the coordinates.

Outlook:
Piece of code which can automatically separate the Laplacian/linear terms from the expansion function

## 5.1 Case 1: symmetry in $\theta$

This case greatly simplifies the linear part in equation (4) since $h$ no longer depends on $\theta$. The details about the solution can be found in the notebook

AppHor_linear-phi.ipynb

Without going too far from the actual solution for the initial guess, this procedures ensures rapid convergence to the actual solution, which is shown below.

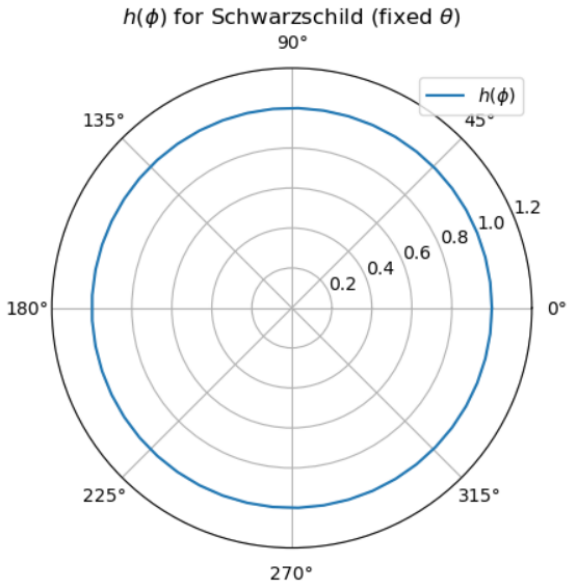


Figure 3: Solution for $h(\phi)$ with the linear solver.

### 5.2 Case 2: symmetry in $\phi$

This case is solved in the same way, except that boundary conditions need to be handled in a different way. Together with the solution, this is explained in the notebook

AppHor_linear-theta.ipynb

This time, the solver is able to find the correct solution for this case, which is shown below.

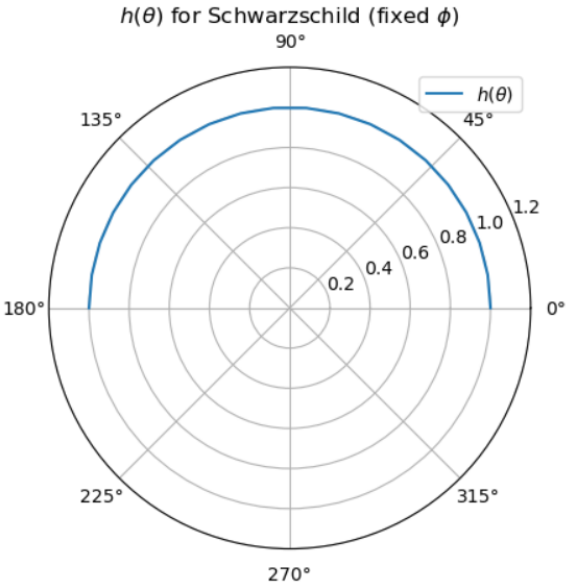


Figure 4: Solution for $h(\theta)$ with the linear solver.

### 5.3 Case 3: general case without assuming symmetries

In this case we combine the methods from the previous two cases into one solver. The main complication comes from the fact that $h$ is now a two-dimensional array, which means it is no longer possible to directly compute a matrix multiplication like $A \cdot h = S(h)$. Instead, we convert $h$ into one long 1D-array using "numpy.ravel()" as well as the right-hand-side $S(h)$ before computing the matrix product. This, however, also complicates the structure of the matrix $A$. The details can be found in the notebook

AppHor_linear-3D.ipynb

We will make some short notes on the current results:

- High grid resolutions make the solver fail to converge
- When the initial condition is set as the actual solution, the solver finds the right solution (this is obviously the most trivial case)
- When the initial condition is off, the solver converges to a solution, but not the right one. This is still to be investigated.

Outlook:
Check if script works for unequal grid dimensions in $\theta$ and $\phi$.
Make script that can automatically separate the linear parts (Laplacian) given the symbolic equation.
Perform a spectral decomposition into spherical harmonics because these are eigenfunctions of the spherical Laplacian.

## References

[1] H.-K. Hui, L.-M. Lin (2024), Revisiting the apparent horizon finding problem with multigrid methods, Chinese University of Hong Kong.