Kerr black hole ergosphere 3D embedding

Alessandro Rovetta

June 3, 2018

1 Analytical embedding

We assume $[x(\theta,\phi), y(\theta,\phi), z(\theta)] := [\rho(\theta)\cos\phi, \rho(\theta)\sin\phi, z(\theta)]$ that respects the Euclidean metric

$$ds^2 = dx^2 + dy^2 + dz^2 = [\ \rho'(\theta)^2 + z'(\theta)^2\ ]\ d\theta^2 + \rho(\theta)^2 d\phi^2 \tag{1.0.1}$$

and we take in exam the hypersurfaces $t = const.\ \wedge\ r = 1 + \sqrt{1 - a^2\cos\theta^2} = r_s^+$ [1], which means considering the ergosphere of our Kerr black hole in a fixed instant, and which leads to

$$dt = 0 \wedge dr_s^+ = \frac{a^2 \sin\theta\cos\theta}{r_s^+ - 1}\ d\theta = c_{11}\ d\theta. \tag{1.0.2}$$

Thus now we can introduce these terms into the line element expressed in Boyer-Lindquist coordinates, obtaining

$$ds^2 = [\bar{g}_{11}\ c_{11}^2 + \bar{g}_{22}]\ d\theta^2 + \bar{g}_{33}\ d\phi^2 \tag{1.0.3}$$

with

$$\bar{g}_{11} = g_{11}(r_s^+) = \frac{\Sigma(r_s^+)}{\Delta(r_s^+)} = \frac{2r_s^+}{a^2\sin^2\theta}$$

$$\bar{g}_{22} = g_{22}(r_s^+) = \Sigma(r_s^+) = 2r_s^+$$

$$\bar{g}_{33} = g_{33}(r_s^+) = 2\sin^2\theta(r_s^+ + a^2\sin^2\theta).$$

So matching the metric we have:

$$\rho(\theta) = \sqrt{\bar{g}_{33}}\ \wedge\ z'(\theta) = \sqrt{\bar{g}_{11}\ c_{11}^2 + \bar{g}_{22} - \rho'(\theta)^2}\ \text{[2]}. \tag{1.0.4}$$

Of course this final equation is not integrable, but we can act this way to avoid the problem: we build an iterative algorithm that prints the function points for a chosen increment, and then interpolate them

[1] using c=G=M=1 i.e. natural units.

[2] the embedding condition is $|\rho'(\theta)| \leq \sqrt{\bar{g}_{11}\ c_{11}^2 + \bar{g}_{22}}$.

with a polynomial function (very easy to integrate).[3] In order to choose the right increment, we can plot $z'(\theta)$ with any free software (like Wolfram Alpha, Maxima etc...) and observe its behaviour[4].

2 Embedding plot for a=0.64

Now let's discuss what I found. The interpolations for the $z'(\theta)$ functions are really good as we can see in the Figure 1 (exploiting the perodicity)

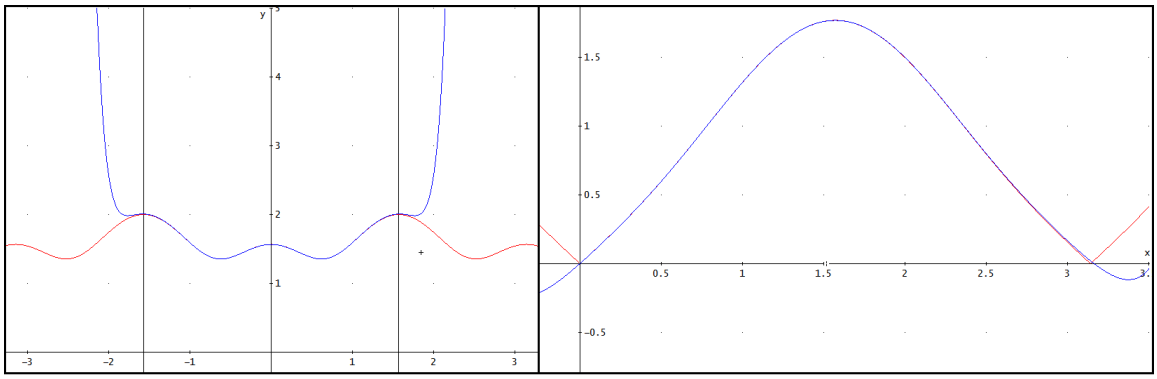

Figure 1: *From the left we have: BH ergosphere and EH $z'(\theta)$ plots in cartesian coordinate. Legend: blue $\leftrightarrow$ interpolation, red $\leftrightarrow$ real function.*

Using the following $z(\theta)$ functions

$$\begin{cases} z(\theta)_{ER} = \sum_{j=0}^{9} A_j \theta^j \\ \\ z(\theta)_{EH} = \sum_{k=0}^{9} B_k \theta^k \end{cases} \tag{2.0.1}$$

$\boldsymbol{A} = [1.5656, 0.0249015^*, -0.6747667, 1.453^*, -3.88, 7.2406667^*, -7.179, 3.866125^*, -1.0885667, 0.126576^*]$

$\boldsymbol{B} = [0.0017668, 0.5224, 0.1376867, -0.21216, 0.42266, -0.3633333, 0.1327271, -0.0201, 0.0004346, 0.0001193]$

where all components marked with * must be multiplied by $Sign(\theta)$, we can finally "draw" a Kerr black hole ergosphere *embedded* in an Euclidean space. But in order to do that, noting that no offset has been set for the $z(\theta)$ coordinate, we must remember that the quantity ds^2 is conserved under each generic diffeomorfphic coordinates change; thus every hypersurface timelike or spacelike tangent vector, must remain as such. This is the reason why we need to set the EH $z(0) = 0.9710162279$: to "keep" the event horizon "symmetricaly inside" the ergosphere.

An other interesting result is that for $\theta = 0$ and $\theta = \pi$ the two $2D$ surfaces are no longer in contact (as we can clearly see in Fig. 3 in the next page).

[3] at the end of the paper I propose an example of program in C++ source code.
[4] I followed the same procedure for the horizon

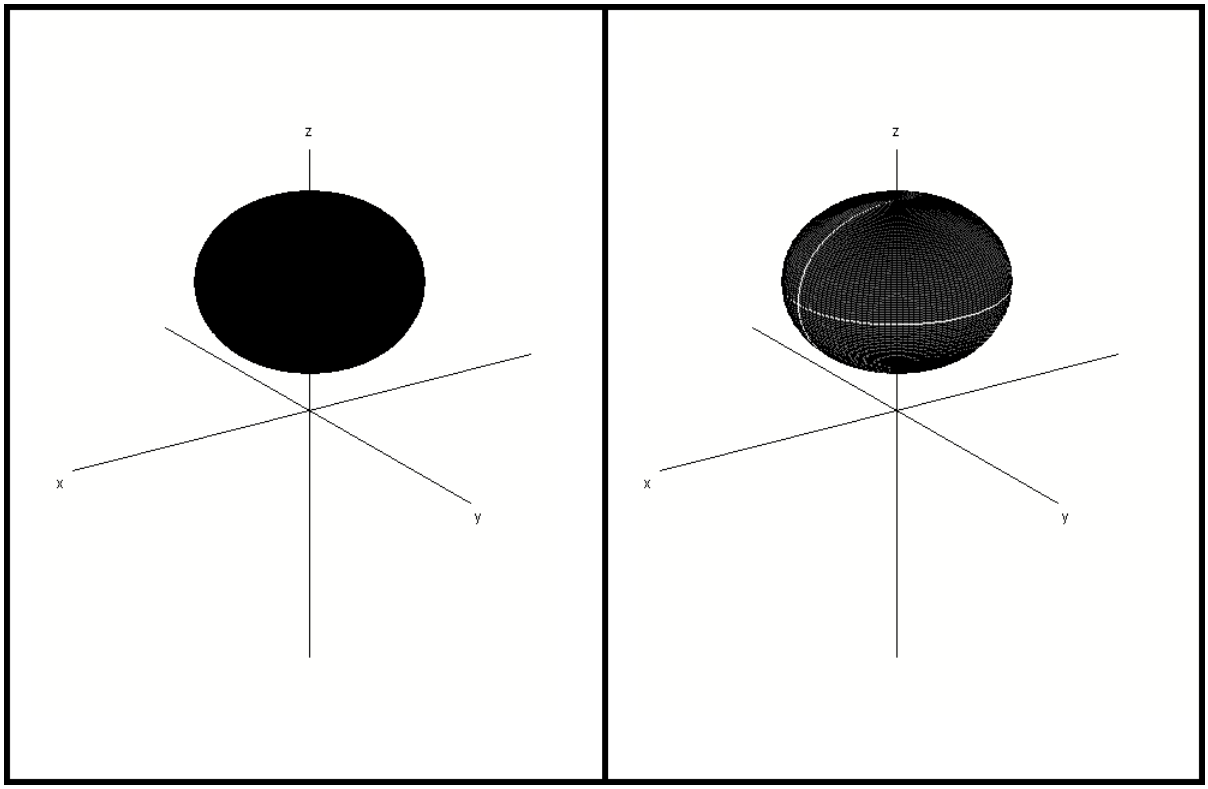

Figure 2: *Black hole event horizon* $3D$ *embedding.*

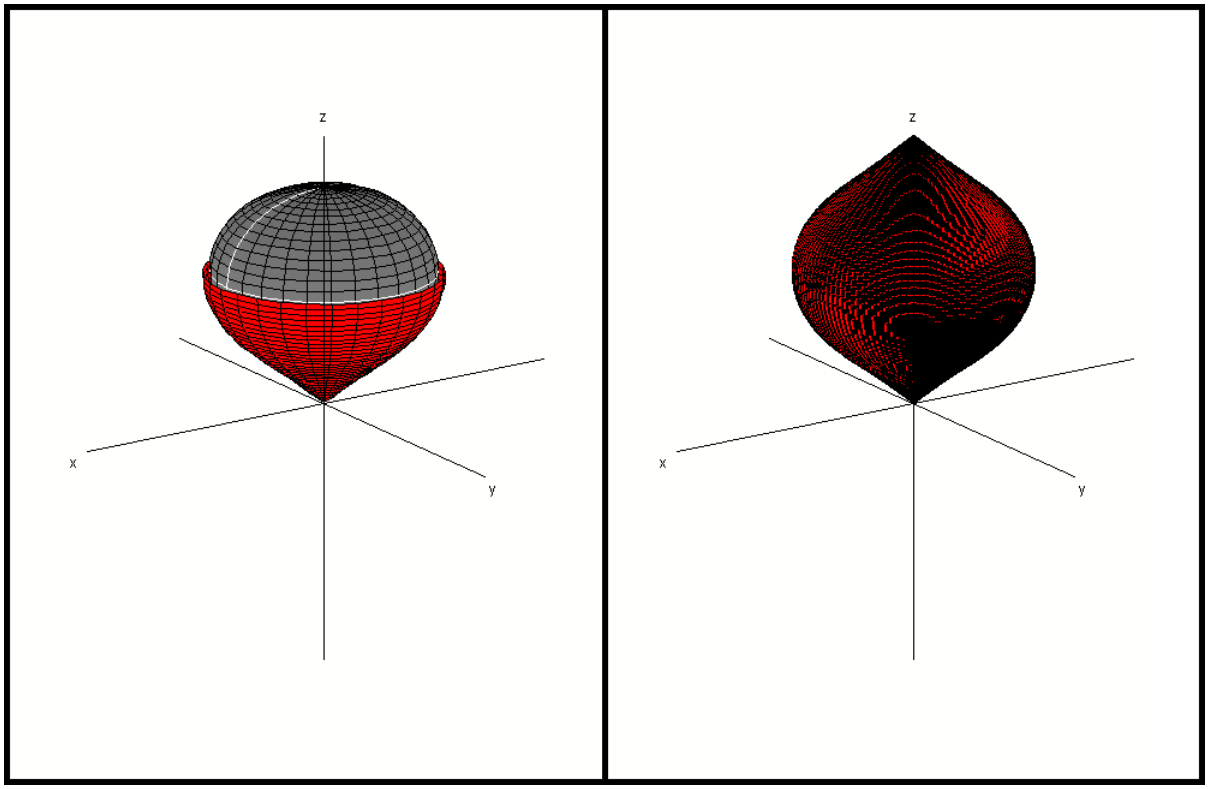

Figure 3: *From the left we have: 1) BH "semi" ergosphere plotted with* $\theta \in [0, \frac{\pi}{2}]$ *and EH plotted with* $\theta \in [0, \pi]$*, 2) BH ergosphere complete* $3D$ *embedding.*

3 Appendix: ergosphere $z'(\theta)$ program C++ source code

```
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <math.h>
#define PI 3.14159265
using namespace std;

int main (){

double a, d, T, z;
int i=1;
T=PI;
cout << "Enter the coefficient alpha: " << endl;
cin >> a; cout << " " << endl;
cout << "Enter the finite increment: " << endl;
cin >> d;
do {
T=T-d;
z=sqrt(2*(sqrt((1 - a*a*cos(i*d)*cos(i*d))*(1 - a*a*cos(i*d)*cos(i*d))*(1 - a*a*cos(i*d)*cos(i*d))) +
a*a*cos(i*d)*cos(i*d)*sqrt(1 - a*a*cos(i*d)*cos(i*d)) + 1)/(1 - a*a*cos(i*d)*cos(i*d)) -
(sqrt(2)*cos(i*d)*abs(sin(i*d))/sin(i*d)*(2*sqrt(1 - a*a*cos(i*d)*cos(i*d))*
(2*a*a*sin(i*d)*sin(i*d) + 1) - 3*a*a*cos(i*d)*cos(i*d) + a*a + 2)/(2*sqrt(1 - a*a*cos(i*d)*cos(i*d))*sqrt(sqrt(1
- a*a*cos(i*d)*cos(i*d)) + a*a*sin(i*d)*sin(i*d) + 1)))*(sqrt(2)*cos(i*d)*abs(sin(i*d))/sin(i*d)*(2*sqrt(1
- a*a*cos(i*d)*cos(i*d))*(2*a*a*sin(i*d)*sin(i*d) + 1) - 3*a*a*cos(i*d)*cos(i*d) + a*a + 2)/(2*sqrt(1 -
a*a*cos(i*d)*cos(i*d))*sqrt(sqrt(1 - a*a*cos(i*d)*cos(i*d)) + a*a*sin(i*d)*sin(i*d) + 1)))); cout << z
<< " " << i*d << endl;
i=i+1;
} while (T>0);

return 0;}
```

3.1 event horizon $z'(\theta)$ program C++ source code

Just change

```
T=2*PI;

z=sqrt(2*(sqrt(1 - 0.64*0.64) + 1) - 0.64*0.64*sin(i*d)*sin(i*d) - (sqrt(2)*cos(i*d)*abs(sin(i*d))/sin(i*d)*sqrt((0.64*(0.64
- 1)*(sqrt(1 - 0.64*0.64) + 1)*sin(i*d)*sin(i*d) - 2*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2))/(0.64*0.64*sin(i*d)*sin(i*d)
- 2*(sqrt(1 - 0.64*0.64) + 1)))*(0.64*0.64*0.64*(0.64 - 1)*(sqrt(1 - 0.64*0.64) + 1)*sin(i*d)*sin(i*d)*sin(i*d)*
sin(i*d) + 4*0.64*(1 - 0.64)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2)*sin(i*d)*sin(i*d) + 4*(sqrt(1 -
0.64*0.64) + 1)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2))/(0.64*0.64*0.64*(0.64 - 1)*(sqrt(1 - 0.64*0.64) +
1)*sin(i*d)*sin(i*d)*sin(i*d)*sin(i*d) + 2*0.64*(1 - 2*0.64)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2)*sin(i*d)*sin(i*d)
+ 4*(sqrt(1 - 0.64*0.64) + 1)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2)))*(sqrt(2)*cos(i*d)*abs(sin(i*d))/(sin(i*d))*sqrt((
0.64*(0.64 - 1)*(sqrt(1 - 0.64*0.64) + 1)*sin(i*d)*sin(i*d) - 2*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 +
2))/(0.64*0.64*sin(i*d)*sin(i*d) - 2*(sqrt(1 - 0.64*0.64) + 1)))*(0.64*0.64*0.64*(0.64 - 1)*(sqrt(1 - 0.64*0.64)
+ 1)*sin(i*d)*sin(i*d)*sin(i*d)*sin(i*d) + 4*0.64*(1 - 0.64)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2)*sin(i*d)*sin(i*d)
+ 4*(sqrt(1 - 0.64*0.64) + 1)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2))/(0.64*0.64*0.64*(0.64 - 1)*(sqrt(1 -
0.64*0.64) + 1)*sin(i*d)*sin(i*d)*sin(i*d)*sin(i*d) + 2*0.64*(1 - 2*0.64)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64
+ 2)*sin(i*d)*sin(i*d) + 4*(sqrt(1 - 0.64*0.64) + 1)*(2*sqrt(1 - 0.64*0.64) - 0.64*0.64 + 2))));
```