THE SHAPE OF ORBITS IN THE SCHWARZSCHILD GEOMETRY

This is a *Mathematica* program to compute and display the shapes of orbits in a Schwarschild geometry. The Schwarzschild radial coordinate is measured in units of M, so that $M=1$ in the following formulae.Throughout the variable $u=1/r$ is used. Here is a list of what you must set to run the program:

(1) The angular momentum: ℓ.
(2) The energy parameter: $\mathcal{E} = (e^2 - 1)/2$.
(3)The starting radius for an orbit which is not bound: *rst.*
(4)The number of orbits to be computed if the orbit is bound: *norbit*.

These are set by editing the definition statements at various places in the program. **You have to make sure these parameters are set so the orbit is classically allowed and doesn't start at a position where the value of the effective potential that is greater than $\mathcal{E}$.**

■ The potential:

The effective potential for radial motion V_{eff} given in (9.28) is here denoted simply by V. To be slightly more general a parameter **signewt** is introduced which multiplies the non–Newtonian $1/r^3$ term in the potential. Set it equal to 1 for general relativity, 0 for a Newtonian $1/r$ potential, and an appropriate value for a Newtonian $1/r$ potential withan additional quadrupole moment term.

```
In[47]:= signewt = 1.

Out[47]= 1.

In[48]:= V[u_, ℓ_] := -u + ℓ^2 u^2 / 2 - signewt ℓ^2 u^3
```

■ Specifying the orbit:

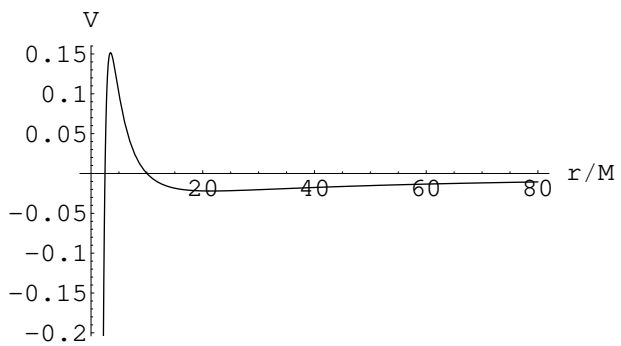
It takes four numbers to specify an orbit: (1) the angular momentum ℓ, (2) the energy parameter $\mathcal{E}$, (3) the starting radius *rst*, and (4) the number of orbits to be calculated, *norbit.*

(1) Specify the **angular momentum ℓ :**

```
In[49]:= ℓ = 5

Out[49]= 5
```

This is a graph of the effective potential vs r/M for this value of ℓ:

```
In[50]:= veff = Plot[V[1 / r, ℓ], {r, 2, 80}, AxesLabel -> {"r/M", "V"}]
```




```
Out[50]= - Graphics -
```

Next the values of the potential at its extrema are computed.

```
In[51]:= dV[u_, ℓ] := -1 + ℓ^2 u  - 3 signewt ℓ^2 u^2
         maxmin = NSolve[dV[ex, ℓ] == 0, ex]
Out[52]= {{ex → 0.0464816}, {ex → 0.286852}}

In[53]:= vmin = V[ex /. maxmin[[1]], ℓ]
Out[53]= -0.0219855

In[54]:= vmax = V[ex /. maxmin[[2]], ℓ]
Out[54]= 0.151615
```

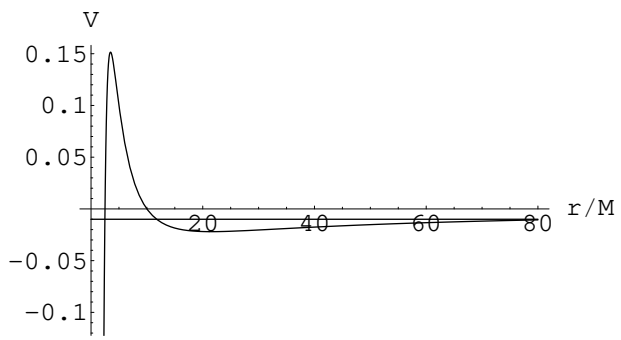
(2) Specify the **energy parameter** $\mathcal{E}$. It must be greater than the minimum value of the potential given above.

```
In[55]:= ℰ = -.01
Out[55]= -0.01
```

Next we plot the effective potential and the energy parameter $\mathcal{E}$ on the same graph:

```
In[56]:= sce := Plot[ℰ, {r, 0, 80}, DisplayFunction → Identity]
```

```
In[57]:= Show[veff, sce, DisplayFunction → $DisplayFunction]
```




```
Out[57]= - Graphics -
```

From this information you can see what kind of orbit it will be — bound orbit, plunge orbit, or scattering orbit.

Now the program evaluates the three turning points associated with this value of the energy parameter:

```
In[58]:= soln = NSolve[V[tp, ℓ] == ℰ, tp]

Out[58]= {{tp → 0.0116597}, {tp → 0.0850696}, {tp → 0.403271}}

In[59]:= tp1 = tp /. soln[[1]]

Out[59]= 0.0116597

In[60]:= tp2 = tp /. soln[[2]]

Out[60]= 0.0850696

In[61]:= tp3 = tp /. soln[[3]]

Out[61]= 0.403271
```

(3) Specify the **starting radius rst.** This together with the energy parameter determines what kind of obrit is computed — bound, asymptotic from and to infinity, plunge, emerging from *r=2M*, etc. depending on where it is set relative to the above turning points. However, only ifthe orbit goes off to infinity is the *value* of *rst* used.

```
In[62]:= rst = 20.

Out[62]= 20.

In[63]:= ust = 1 / rst

Out[63]= 0.05
```

(4) Finally specify the **number of orbits, norbit ,** if the orbit is bound.

```
In[64]:= norbit = 3

Out[64]= 3
```

■ Computing the Orbit

We are now goint to integrate $\{(\ell/2^{1/2})[\mathcal{E}-V[u,\ell]]^{(-1/2)}$to find the orbit in the form ϕ(r). The integrand diverges near the turning points, so we specify a parameter **eps** which determines how close the numerical integration comes to the turning points.

```
In[65]:= eps = .00000001

Out[65]= 1. × 10^-8
```

Next the program picks the appropriate radii to start and end the integration of the orbit given the energy parameter and starting radius. There are four kinds of orbits.
(a) Bound obits which start at the outer turning point.
(b) Orbits which come in from infinity and go out again. These start at *rst.*
(c) Orbits which start close to *r=2M* and fall back to it.
(d) Plunge orbits which start at rst and end at *r=2M*. For bound the starting radii are the

Tests for bound orbits:

```
In[66]:= testa = (ℰ < 0) && (ust < tp2)

Out[66]= True

In[67]:= If[testa, u1 = tp1 (1 + eps); u2 = tp2 (1 - eps)]

Out[67]= 0.0850696
```

Tests for orbits coming from and going to infinity:

```
In[68]:= testb = (ℰ > 0) && (ℰ < vmax) && (ust < tp2)

Out[68]= False

In[69]:= If[testb, u1 = ust;  u2 = tp2 (1 - eps)]
```

Tests for orbits which start from *r=2M* and go back to it:

```
In[70]:= testc = (ℰ < vmax) && ( ust > tp3)

Out[70]= False

In[71]:= If[testc, u1 = .5; u2 = tp3 (1 + eps)]
```

Tests for plunge orbits coming from infinity going to *r=2M*:

```
In[72]:= testd = (ℰ > vmax)

Out[72]= False

In[73]:= If[testd, u1 = ust; u2 = .5 (1 - eps)]
```

If the parameters are set correctly only one of the four options should be ‘‘true’’ andthe other three should be ‘‘false’’. The values of *u1* and *u2* are displayed below:

```
In[74]:= u1

Out[74]= 0.0116597

In[75]:= u2

Out[75]= 0.0850696
```

A number of functions are defined, the chief of which is the angle **theta** which is the angle ϕ swept out from the innermost turning point. This is given by the following numerical integral:

```
In[76]:= theta[u_, ε_, ℓ_, u1_] := NIntegrate[(ℓ / 2^(1 / 2)) (ε - V[w, ℓ])^(-1 / 2), {w, u1, u}]
```

delphi is the total angle ϕ swept out between two endpoints of the orbit:

```
In[77]:= delphi = theta[u2, ε, ℓ, u1]

Out[77]= 3.73604
```

z is a parameter which varies from 0 to *norbit.*

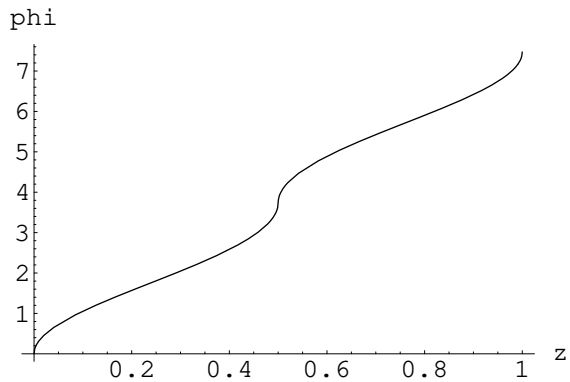
```
In[78]:= n[z_] := IntegerPart[z]

In[79]:= zf[z_] := FractionalPart[z]

In[80]:= ua[z_] := u1 (1 - 2 zf[z]) + u2  2 zf[z]

In[81]:= ub[z_] := u1 (2 zf[z] - 1)  + 2 u2 (1 - zf[z])

In[82]:= u[z_] := If[zf[z] < .5, ua[z], ub[z]]

In[83]:= phia[z_] := 2 (n [z]) delphi + theta[u[z], ε, ℓ, u1]

In[84]:= phib[z_] := 2 (n [z] + 1) delphi - theta[u[z], ε, ℓ, u1]

In[85]:= accphi[z_] := If[zf[z] < .5, phia[z], phib[z]]

In[86]:= x[z_] := Cos[accphi[z]] / u[z]

In[87]:= y[z_] := Sin[accphi[z]] / u[z]
```

■ Displaying the Results:

accphi is the accumulated angle as a function of position along the orbit, as measured by the parameter z defined above to vary from 0 to norbit, so that *z=1* is the end of the first orbit, *z=2* is the end of the second, etc.

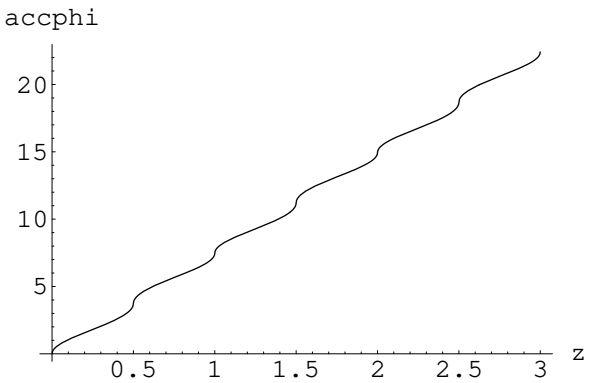
```
In[88]:= If[testa, norbit = norbit, norbit = 1]

Out[88]= 3

In[89]:= If[testd, norbit = .5]
```

In[90]:= **If[norbit > 1, Plot[accphi[z], {z, 0, 1}, AxesLabel -> {"z", "phi"}]]**



Out[90]= **-** Graphics **-**

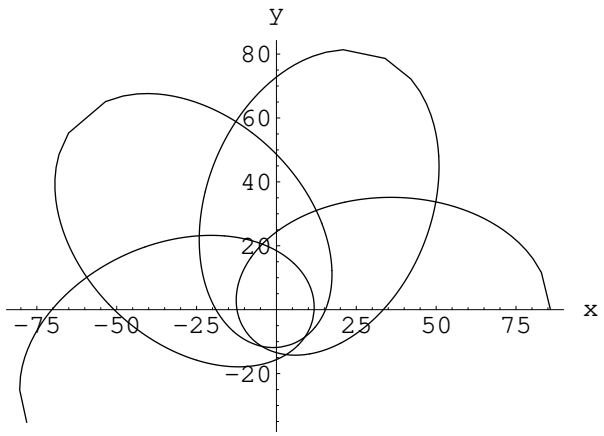
In[91]:= **Plot[accphi[z], {z, 0, norbit}, AxesLabel -> {"z", "accphi"}]**



Out[91]= **-** Graphics **-**

Calculating and plotting the orbit: Two parameters specify how accurately the orbit is calculated: **PlotDivision** specifies the number of plot points, **MaxBend** specifies the maximum bending angle permitted. For shorter calculations and rougher orbits decrease PlotDivision and increase MaxBend. For longer calculations and more accurate orbits increase **PlotDivision** and decrease **MaxBend.**

```
In[92]:= graph = ParametricPlot[{x[t], y[t]}, {t, 0, norbit}, MaxBend -> .1,
            PlotDivision -> 50, AspectRatio → Automatic, AxesLabel -> {"x", "y"}]
```




```
Out[92]= - Graphics -
```

If you want to output a copy of the orbit as an eps file called orbit.eps you can include the statment:
Display["orbit.eps", graph, "EPS"].