

# Doing Full Spectrum Physics

## Three Aspects of Geometry

### Misner, Thorne & Wheeler, *Gravitation*, Section 8.3

**David Park, [djmp@earthlink.net](mailto:djmp@earthlink.net)**

Misner, Thorne and Wheeler teach that we will have full power to do and communicate geometry and physics if this power "can be exercised in three ways: in pictures, in abstract notation and in component notation." With *Mathematica* and a proper suite of packages to close the gap between *Mathematica*'s basic power and the practice of mathematical physics we can do this - and better. We can also have active mathematical objects, incorporate units, and we can even bring solutions and diagrams alive with animation. This is what I call being able to do *full spectrum physics* within a single computer platform.

## Initiation

The following are a suite of *Mathematica* packages that allow us to conveniently do physics.

```
In[1]:= Needs["Miscellaneous`PhysicalConstants`"]
        Needs["Units`ExtendUnits`"]
        Needs["TensorCalculus4`Tensorial`"]
        Needs["DrawGraphics`DrawingMaster`"]
        Needs["Cardano3`ComplexGraphics`"]
```

We are going to derive and solve the equations for planetary motion in polar coordinates. The following statements establish conditions for the *Tensorial* tensor calculus package. We define tensor shortcuts for the basis vectors  $\mathbf{e}$ , the velocity vector  $\mathbf{v}$ , and the coordinate positions  $\mathbf{x}$ . Since we will be using an orthonormal polar frame we use red indices to remind us that we are not working in a coordinate basis. Finally, we declare a 2-dimensional system with base indices corresponding to the coordinate symbols.

```
In[6]:= DefineTensorShortcuts[{{e, v, x}, 1}]
        DeclareIndexFlavor[{red, Red}]
        DeclareBaseIndices[{r, phi}]
```

## Abstract, Pictorial and Component Form of Keplerian Orbit Physics

We have the following general symbolic expression for the velocity of a planet, which we will take to be in a Keplerian orbit,

```
In[9]:= v == vu[a] ed[a] // ToFlavor[red]
      eqn[8, 1] = % // EinsteinSum[]
```

```
Out[9]= v == e_a v^a
```

```
Out[10]= v == e_r v^r + e_phi v^phi
```

where the red flavor indicates we are working in an orthonormal frame where the  $\mathbf{e}$  are unit vectors.

The acceleration is obtained by taking the total derivative with respect to time.

```
In[11]:= TotalD[eqn[8, 1], t]
      eqn[8, 2] = % /. TotalD[v, t] -> a
```

```
Out[11]=  $\frac{dv}{dt} = v^r \frac{de_r}{dt} + v^\phi \frac{de_\phi}{dt} + e_r \frac{dv^r}{dt} + e_\phi \frac{dv^\phi}{dt}$ 
```

```
Out[12]=  $a = v^r \frac{de_r}{dt} + v^\phi \frac{de_\phi}{dt} + e_r \frac{dv^r}{dt} + e_\phi \frac{dv^\phi}{dt}$ 
```

A diagram for the planet's motion is as follows. (In the notebook evaluate the thin closed cell.) Normally, in a notebook, the following input cell would be closed because the reader would not be interested in the code that produces a geometrical diagram. Here the cell is left open to illustrate how custom geometrical diagrams can easily be produced with the help of the DrawGraphics package. Any nice geometrical diagram requires detailed specification. The advantage of DrawGraphics is that all of the elements of the diagram, curves, lines, arrows and labels can be combined in a single graphics statement. The various elements are simply drawn, one after another, as with conventional drawing programs.

For this diagram, the curve representing a portion of the orbit was produced freehand by clicking off a set of points on the diagram and then converting them to a smooth **Line** primitive with the **SplineToLine** routine. The two points on the orbit were also simply clicked off the diagram. The radial unit vectors were obtained by normalizing the two orbit points. Since it is easy to remember how to rotate vectors in the complex plane two routines, **ToComplex** and **ToCoordinates**, from the Cardano3 package were used to produce the  $\phi$  basis vectors. The DrawGraphics **AngleArc** routine was used to produce a label for the  $\phi$  angle with a curved arrow. The **ColorMix** routine was used to obtain a pale version of the basis vectors.

```
In[13]:= Module[
  {pts, orbit, sun, ptA, ptB, avector, vvector, erA, ephiA, erB, ephiB, drawarrow},

  pts = {{2.27893, 7.1413}, {4.17476, 6.90056}, {5.85995, 6.53945},
    {7.78588, 6.08806}, {9.53125, 5.60658}, {11.3669, 4.85426}, {13.1725, 3.8913}};
  orbit = SplineToLine[pts, Bezier, 20];
  drawvector[start_, vector_] := Arrow[start, start + vector,
    HeadScaling -> Relative, HeadCenter -> 0.5, HeadLength -> 0.2, HeadWidth -> 0.75];
  sun = {0, 0};
```

```

ptA = {8.17142, 5.92416};
ptB = {3.61412, 6.98325};
erA = 2 # / Sqrt[#.#] &[ptA];
eφA = ToCoordinates[Exp[i π / 2] ToComplex[erA]];
erB = 2 # / Sqrt[#.#] &[ptB];
eφB = ToCoordinates[Exp[i π / 2] ToComplex[erB]];
avector = -erA;
vvector = 3 # / Sqrt[#.#] &[1.05 ptB - ptA];

Draw2D[
{orbit,
  Gray, Line[{sun, ptA}], Line[{sun, ptB}],
  Line[{sun, {5, 0}}],
  AngleArc[sun, {0, ArcTan@@ptA}, 3, StyleForm["φ", FontColor → Black]],

  (* Unit basis vectors *)
  VenetianRed,
  drawvector[ptA, erA],
  drawvector[ptA, eφA],
  drawvector[ptB, erB],
  drawvector[ptB, eφB],
  ColorMix[VenetianRed, White][0.7],
  drawvector[ptB, erA],
  drawvector[ptB, eφA],
  Chartreuse,
  drawvector[ptB + erA, erB - erA],
  drawvector[ptB + eφA, eφB - eφA],

  (* Acceleration and velocity vectors *)
  Black, AbsoluteThickness[2],
  drawvector[ptA, avector],
  drawvector[ptA, vvector],
  AbsoluteThickness[1],

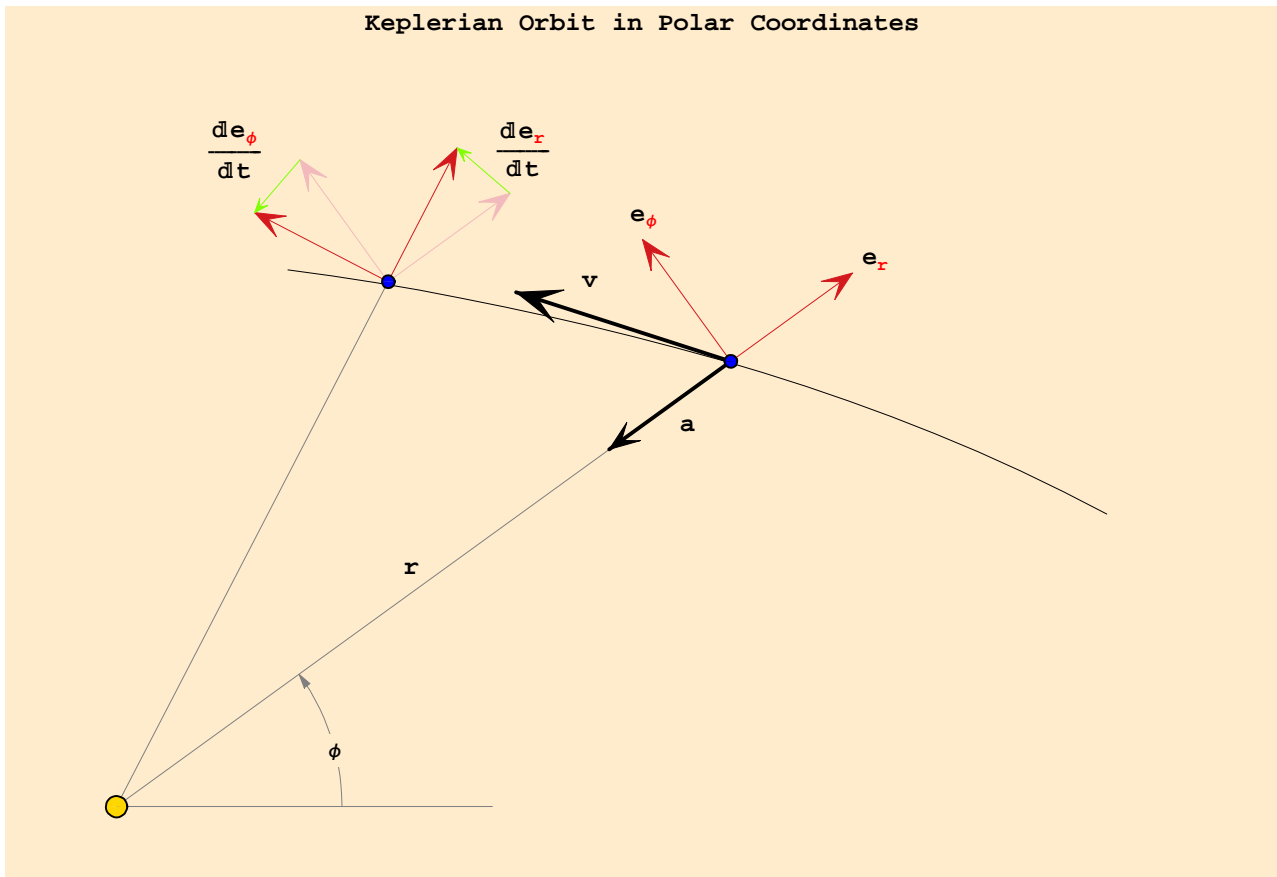
  (* Points *)
  CirclePoint[sun, 5, Black, Gold],
  CirclePoint[ptA, 3, Black, Blue],
  CirclePoint[ptB, 3, Black, Blue],

  (* Text labels *)
  Black,
  Text["r", 1/2 ptA, {1, -1}],
  Text["a", {7.59905, 5.08459}],
  Text["v", {6.29826, 7.00155}],
  Text[ed[red@r], {10.0979, 7.30964}],
  Text[ed[red@φ], {7.01712, 7.89157}],
  Text[TotalD[ed[red@r], t], {5.40824, 8.74736}],
  Text[TotalD[ed[red@φ], t], {1.57431, 8.78159}]],

AspectRatio → Automatic,

```

```
PlotRange → {{-1, 15}, {-1, 10}},
PlotLabel → "Keplerian Orbit in Polar Coordinates",
Background → BlendedAlmond,
ImageSize → 600];
```



The orthonormal basis vectors are shown in red. As the planet moves the basis vectors rotate and the rate of rotation is just the rate of change of  $\phi$ . This is, in fact, the advantage of using an orthonormal frame. We can express this by the following rules. We also write the rules for the red velocity components in terms of the coordinates.

```
In[14]:= eqn[8, 3] = {TotalD[ed[red@r], t] → ω ed[red@φ], TotalD[ed[red@φ], t] → -ω ed[red@r]}
vrule = ω → TotalD[xu[φ], t]
vrules = {vu[red@r] → TotalD[xu[r], t], vu[red@φ] → xu[r] TotalD[xu[φ], t]}
```

```
Out[14]= {  $\frac{de_r}{dt} \rightarrow \omega e_\phi$ ,  $\frac{de_\phi}{dt} \rightarrow -\omega e_r$  }
```

```
Out[15]=  $\omega \rightarrow \frac{d\phi}{dt}$ 
```

```
Out[16]= {  $v^r \rightarrow \frac{dx^r}{dt}$ ,  $v^\phi \rightarrow x^r \frac{d\phi}{dt}$  }
```

We then use these rules to substitute into equation 8.2, rearrange, and substitute coordinate expressions to obtain the components of the acceleration.

```

In[17]:= eqn[8, 2]
Print["Substitute the rate of change of the basis vectors"]
%% /. eqn[8, 3]
Print["Collect terms in the two directions"]
eqn[8, 4] = MapAt[Collect[#, ed[red@a] // ToArrayValues[]] &, %, 2]
Print["Substitute the velocity and  $\omega$  expressions"]
equation1 = %% /. vrules /.  $\omega$ rule
Print["Substitute coordinates as functions of t"]
eqn[8, 4] = %% // UseCoordinates[{r[t],  $\phi$ [t]}]

```

$$\text{Out[17]} = \mathbf{a} = v^r \frac{d\mathbf{e}_r}{dt} + v^\phi \frac{d\mathbf{e}_\phi}{dt} + \mathbf{e}_r \frac{dv^r}{dt} + \mathbf{e}_\phi \frac{dv^\phi}{dt}$$

Substitute the rate of change of the basis vectors

$$\text{Out[19]} = \mathbf{a} = \omega \mathbf{e}_\phi v^r - \omega \mathbf{e}_r v^\phi + \mathbf{e}_r \frac{dv^r}{dt} + \mathbf{e}_\phi \frac{dv^\phi}{dt}$$

Collect terms in the two directions

$$\text{Out[21]} = \mathbf{a} = \mathbf{e}_r \left( -\omega v^\phi + \frac{dv^r}{dt} \right) + \mathbf{e}_\phi \left( \omega v^r + \frac{dv^\phi}{dt} \right)$$

Substitute the velocity and  $\omega$  expressions

$$\text{Out[23]} = \mathbf{a} = \mathbf{e}_r \left( \frac{d^2 x^r}{dt^2} - x^r \left( \frac{dx^\phi}{dt} \right)^2 \right) + \mathbf{e}_\phi \left( 2 \frac{dx^r}{dt} \frac{dx^\phi}{dt} + x^r \frac{d^2 x^\phi}{dt^2} \right)$$

Substitute coordinates as functions of t

$$\text{Out[25]} = \mathbf{a} = \mathbf{e}_r (-r[t] \phi'[t]^2 + r''[t]) + \mathbf{e}_\phi (2 r'[t] \phi'[t] + r[t] \phi''[t])$$

This is equivalent to the MTW equation 8.4. MTW use the following equivalent expression for the  $\phi$  component. **HoldOp** is a *Tensorial* command that allows us to delay certain operations that would otherwise be automatically performed.

```

In[26]:= D[r[t]  $\phi'$ [t], t] + r'[t]  $\phi'$ [t] // HoldOp[D]
% // ReleaseHold

```

```

Out[26]=  $\partial_t (r[t] \phi'[t]) + r'[t] \phi'[t]$ 

```

```

Out[27]=  $2 r'[t] \phi'[t] + r[t] \phi''[t]$ 

```

Why don't we go further and actually solve the equations and graphically represent the solutions? We do this in the next section.

## Units, Solutions and Graphical Representations

Let's solve the orbit equations for the planet earth. We write the acceleration on the rhs and then derive the differential equations for  $\mathbf{r}$  and  $\phi$ .

```

In[28]:= Clear[r, ϕ]
SetAttributes[{AU, Year}, Constant]
equation1
Print["Write acceleration on lhs for gravitational force."]
%% /. a → -ed[red@r] GM/xu[r]^2
Print["Rearrange and collect terms"]
# - First[%%] & /@ %% // Reverse;
step1 = MapAt[Collect[#, ed[red@a] // EinsteinArray[]] &, %, 1]
Print["Each of the components must be zero"]
Coefficient[First@step1, ed[red@a] // EinsteinArray[]] == {0, 0} // Thread
Print["Convert to Mathematica differential
equations by substituting coordinate expressions."]
diffeqns0 = %% // UseCoordinates[{r[t], ϕ[t]}]

```

$$\text{Out}[30]= \mathbf{a} = e_{\mathbf{r}} \left( \frac{d^2 \mathbf{x}^r}{dt dt} - \mathbf{x}^r \left( \frac{d\mathbf{x}^\phi}{dt} \right)^2 \right) + e_{\phi} \left( 2 \frac{d\mathbf{x}^r}{dt} \frac{d\mathbf{x}^\phi}{dt} + \mathbf{x}^r \frac{d^2 \mathbf{x}^\phi}{dt dt} \right)$$

Write acceleration on lhs for gravitational force.

$$\text{Out}[32]= -\frac{GM e_{\mathbf{r}}}{(\mathbf{x}^r)^2} = e_{\mathbf{r}} \left( \frac{d^2 \mathbf{x}^r}{dt dt} - \mathbf{x}^r \left( \frac{d\mathbf{x}^\phi}{dt} \right)^2 \right) + e_{\phi} \left( 2 \frac{d\mathbf{x}^r}{dt} \frac{d\mathbf{x}^\phi}{dt} + \mathbf{x}^r \frac{d^2 \mathbf{x}^\phi}{dt dt} \right)$$

Rearrange and collect terms

$$\text{Out}[35]= e_{\mathbf{r}} \left( \frac{GM}{(\mathbf{x}^r)^2} + \frac{d^2 \mathbf{x}^r}{dt dt} - \mathbf{x}^r \left( \frac{d\mathbf{x}^\phi}{dt} \right)^2 \right) + e_{\phi} \left( 2 \frac{d\mathbf{x}^r}{dt} \frac{d\mathbf{x}^\phi}{dt} + \mathbf{x}^r \frac{d^2 \mathbf{x}^\phi}{dt dt} \right) = 0$$

Each of the components must be zero

$$\text{Out}[37]= \left\{ \frac{GM}{(\mathbf{x}^r)^2} + \frac{d^2 \mathbf{x}^r}{dt dt} - \mathbf{x}^r \left( \frac{d\mathbf{x}^\phi}{dt} \right)^2 = 0, 2 \frac{d\mathbf{x}^r}{dt} \frac{d\mathbf{x}^\phi}{dt} + \mathbf{x}^r \frac{d^2 \mathbf{x}^\phi}{dt dt} = 0 \right\}$$

Convert to Mathematica differential equations by substituting coordinate expressions.

$$\text{Out}[39]= \left\{ \frac{GM}{r[t]^2} - r[t] \phi'[t]^2 + r''[t] = 0, 2 r'[t] \phi'[t] + r[t] \phi''[t] = 0 \right\}$$

We then need to append the initial conditions and substitute the gravitational constant and the solar mass..

```

In[40]:= {diffeqns0, r[0] == r0, r'[0] == r0dot, ϕ[0] == ϕ0, ϕ'[0] == ϕ0dot} // Flatten;
diffeqns1 = %% /. {G → GravitationalConstant, M → SolarMass}

```

$$\text{Out}[41]= \left\{ \frac{6.673 \times 10^{-11} \text{ Meter}^2 \text{ Newton SolarMass}}{\text{Kilogram}^2 r[t]^2} - r[t] \phi'[t]^2 + r''[t] = 0, \right. \\ \left. 2 r'[t] \phi'[t] + r[t] \phi''[t] = 0, r[0] == r0, r'[0] == r0dot, \phi[0] == \phi0, \phi'[0] == \phi0dot \right\}$$

We set up the data for aphelion of the earth. We calculate the rate of change of the angle at aphelion by using the aphelion rotational velocity and aphelion distance.

```
In[42]:=  $\omega = v / r$ 
          % /. {v → 29.2944 Kilo Meter / Second, r → 1.0167 AU}
          % // ToUnitRule[1 / Day]
```

```
Out[42]=  $\omega = \frac{v}{r}$ 
```

```
Out[43]=  $\omega = \frac{28.8132 \text{ Kilo Meter}}{\text{AU Second}}$ 
```

```
Out[44]=  $\omega = \frac{0.016641}{\text{Day}}$ 
```

We then establish the data for the initial conditions as a set of rules. The units are astronomical units **AU** and days. We could have entered the data using any equivalent units.

```
In[45]:= (data = {r0 → 1.0167 AU, r0dot → 0,  $\phi$ 0 → 0,  $\phi$ 0dot →  $\frac{0.016641}{\text{Day}}$ }) // TableForm
```

```
Out[45]//TableForm=
  r0 → 1.0167 AU
  r0dot → 0
   $\phi$ 0 → 0
   $\phi$ 0dot →  $\frac{0.016641}{\text{Day}}$ 
```

We will want to 'deunitize' the equations to solve them numerically and the best method is to set up reduced units where **G == AU == Day == 1**. This is done with the **ExtendUnits** package, which allows the establishment of reduced unit systems.

```
In[46]:= SetupReducedUnits[{GravitationalConstant, AU, Day}, {Ampere, Kelvin}]
```

```
Out[46]= {Kilogram →  $1.4879 \times 10^{-34}$ , Meter →  $6.68459 \times 10^{-12}$ , Second → 0.0000115741}
```

We then obtain the differential equations in 'geometrical' form.

```
In[47]:= Clear[r,  $\phi$ ]
          diffeqns2 = diffeqns1 /. data // ReducedUnits
```

```
Out[48]= {  $\frac{0.000295957}{r[t]^2} - r[t] \phi'[t]^2 + r''[t] = 0, 2 r'[t] \phi'[t] + r[t] \phi''[t] = 0,$ 
          r[0] == 1.0167, r'[0] == 0,  $\phi$ [0] == 0,  $\phi'$ [0] == 0.016641 }
```

To numerically solve for an orbit we can use a **StoppingTest** in **NDSolve** to detect the completion of  $2\pi$  radians. Then we can extract the **period** (in days) from the returned **InterpolatingFunction**.

```

In[49]:= Clear[r,  $\phi$ ]
First@NDSolve[diffeqns2, {r,  $\phi$ }, {t, 0, 366}, StoppingTest  $\Rightarrow$  ( $\phi[t] \geq 2\pi$ )]
{r[t_],  $\phi[t_]$ } = {r[t],  $\phi[t]$ } /. %
period = Part[Head[r[t]], 1, 1, 2]

Out[50]= {r  $\rightarrow$  InterpolatingFunction[{{0., 365.256}}, <>],
 $\phi$   $\rightarrow$  InterpolatingFunction[{{0., 365.256}}, <>]}

Out[51]= {InterpolatingFunction[{{0., 365.256}}, <>][t],
InterpolatingFunction[{{0., 365.256}}, <>][t]}

Out[52]= 365.256

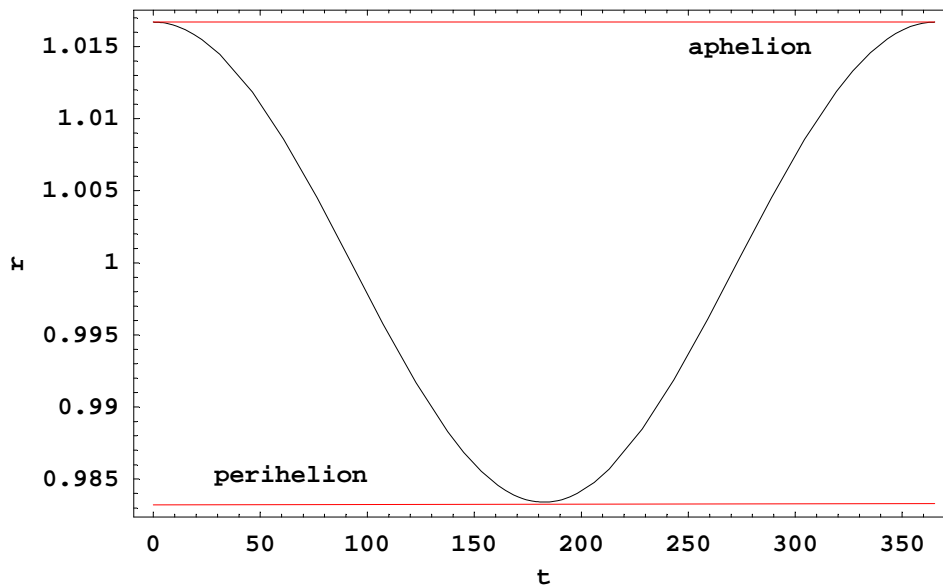
```

The following plot shows the variation of earth's radial distance with time. The aphelion and perihelion radii are marked in red.

```

In[53]:= Draw2D[
  {Draw[r[t], {t, 0, period}],
  Red, Line[{{0, 0.9832}, {period, 0.9832}}],
  Line[{{0, 1.0167}, {period, 1.0167}}],
  Black, Text["perihelion", {64.3781, 0.98528}],
  Text["aphelion", {279.088, 1.01493}],
  Frame  $\rightarrow$  True,
  FrameLabel  $\rightarrow$  {t, r},
  ImageSize  $\rightarrow$  450];

```



The perihelion velocity can be calculated as follows.

```
In[54]:=  $\phi'$ [period / 2] R[aphelion]  
ReducedEquivalent[%, 1, 1 / Day]  
% /. R[aphelion] → 0.9833 AU  
% // ToUnit[Kilo Meter / Second]
```

```
Out[54]= 0.0177871 R[aphelion]
```

```
Out[55]=  $\frac{0.0177871 \text{ R[aphelion]}}{\text{Day}}$ 
```

```
Out[56]=  $\frac{0.0174901 \text{ AU}}{\text{Day}}$ 
```

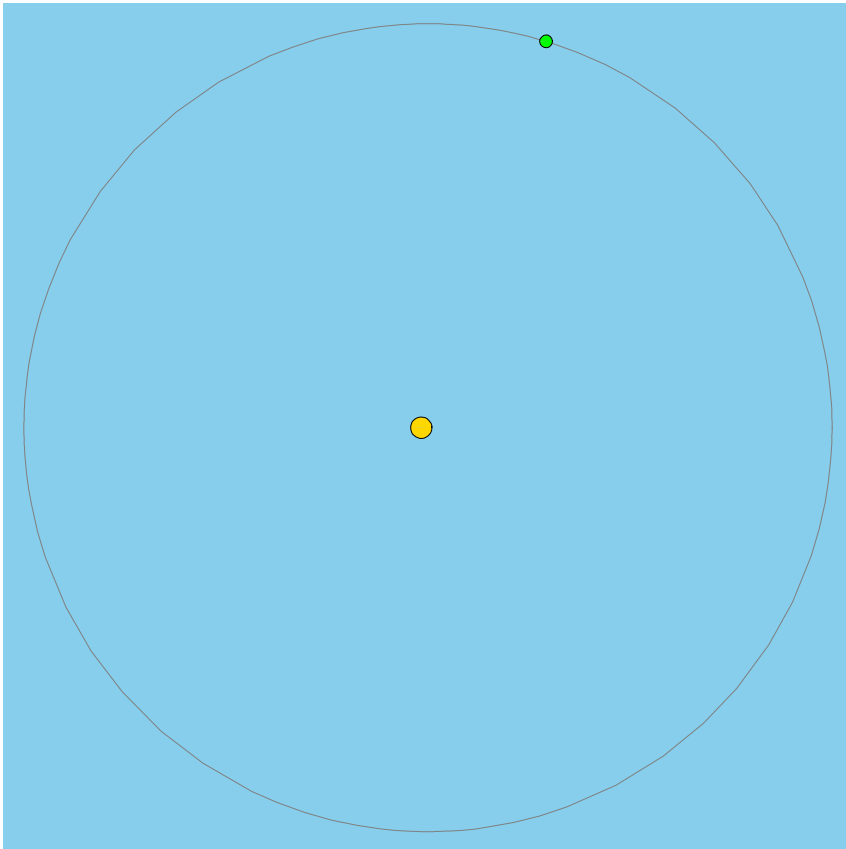
```
Out[57]=  $\frac{30.2833 \text{ Kilo Meter}}{\text{Second}}$ 
```

An animation is easily produced for the orbital motion. The following defines one frame for the animation and displays a test frame at 75 days from aphelion.

```

In[58]:= frame[t_] :=
  Draw2D[
    {CirclePoint[{0, 0}, 5, Black, Gold],
     Gray,
     ParametricDraw[r[s] {Cos[φ[s]], Sin[φ[s]]}, {s, 0, period}],
     CirclePoint[r[t] {Cos[φ[t]], Sin[φ[t]]}, 3, Black, Green]},
    AspectRatio → Automatic,
    Background → SkyBlue,
    ImageSize → 400];
frame[75];

```



The following code will actuate the animation. (The animation itself is not shown in the pdf document.) The statements after the **Animate** statement serve to select and close up the graphics cells and then start the animation. These statements were actually clicked in from a button on the **DrawGraphics** palette. The animation would be more interesting with a more elliptical orbit.

```

Animate[frame[t], {t, 0, del = period / 200; period - del, del}]
SelectionMove[EvaluationNotebook[], All, GeneratedCell]
FrontEndTokenExecute["OpenCloseGroup"]; Pause[0.5];
FrontEndExecute[{FrontEnd`SelectionAnimate[
  200, AnimationDisplayTime → 0.05, AnimationDirection → Forward}]]

```

We might ask the question: What is the maximum initial 'aphelion' velocity that will produce a closed orbit? We write an expression for the energy and transform it to coordinate expressions.

```
In[60]:= Clear[r, ϕ]
SetAttributes[AU, Constant]
ε = 1/2 m (vu[red@r]^2 + vu[red@ϕ]^2) - G M m / xu[r]
% /. vrules
step1 = % // UseCoordinates[{r[t], ϕ[t]}]
```

$$\text{Out[62]} = \varepsilon = \frac{1}{2} m \left( (\mathbf{v}^r)^2 + (\mathbf{v}^\phi)^2 \right) - \frac{G m M}{x^r}$$

$$\text{Out[63]} = \varepsilon = -\frac{G m M}{x^r} + \frac{1}{2} m \left( \left( \frac{d\mathbf{x}^r}{dt} \right)^2 + (x^r)^2 \left( \frac{d\mathbf{x}^\phi}{dt} \right)^2 \right)$$

$$\text{Out[64]} = \varepsilon = -\frac{G m M}{r[t]} + \frac{1}{2} m (r'[t]^2 + r[t]^2 \phi'[t]^2)$$

The energy must be negative for a closed orbit. The maximum value of  $\phi'[t]$  is obtained by setting the rhs expression to zero.

```
In[65]:= Part[step1, 2] == 0
Solve[%, ϕ'[t]] [[2,1]] /. t -> 0
% /. {G -> GravitationalConstant, M -> SolarMass, r[0] -> r0, r'[0] -> r0dot} /. data
% // ToUnitRule[Radian/Day]
```

$$\text{Out[65]} = -\frac{G m M}{r[t]} + \frac{1}{2} m (r'[t]^2 + r[t]^2 \phi'[t]^2) == 0$$

$$\text{Out[66]} = \phi'[0] \rightarrow \frac{\sqrt{\frac{2 G M}{x[0]} - r'[0]^2}}{r[0]}$$

$$\text{Out[67]} = \phi'[0] \rightarrow \frac{0.000011269 \sqrt{\frac{\text{Meter}^2 \text{Newton SolarMass}}{\text{AU Kilogram}^2}}}{\text{AU}}$$

$$\text{Out[68]} = \phi'[0] \rightarrow \frac{0.0237323 \text{ Radian}}{\text{Day}}$$

The earth's aphelion velocity is well below this.

## Full Spectrum Physics

To do real physics on a computer we want to write equations the way that physicists write them, which is mostly in tensor notation. We want to be able to draw geometric diagrams the way physicists and geometers draw them. We want to be able to deal with real data with real units. It takes considerable extra programming on the part of users to close the *Mathematica* gap. The **Tensorial**, **DrawGraphics**, **ExtendUnits** and **Cardano3** packages go a very long way in closing the gap. They allow students and researchers to think much more in terms of physics and mathematics and much less in terms of programming. They allow the thinking and results to be presented in a clear fashion in the language of physicists.