Principles of Linear Algebra With *Maple*™
Rolling an Ellipse Along a Curve

Kenneth Shiskowski and Karl Frinkle

Contents

Chapter 1

Rolling an Ellipse Along a Curve

1.1 The Setup

In Section 7.2 of *Principles of Linear Algebra With* Maple^{TM}, we discussed the topic of "rolling" a circle along a curve. Of course, the rolling was in quotes due to the fact that we were really just sliding the circle along the curve. The animation have the appearance of a rolling circle, but this was due only to the fact a circle is a perfectly symmetric object, and no matter which way it is rotated, there is no apparent change in orientation. So what happens if we pick an object which is not as symmetric? Take, for instance, an ellipse. Clearly if an ellipse is rotated through most angles, it will be obvious to an observer that the ellipse was indeed rotated. The only two angles for which this is not the case is $\theta = \pi$ and $\theta = 2\pi$. The question then becomes: what additionally must be done to rotate an ellipse versus a circle?

To determine some of the math required to rotate an ellipse, we will start with a simple setup, that of rolling the ellipse along a horizontal line, similar to rolling a football, along a level surface, end over end. We define an ellipse with center at (x_c, y_c) parametrically as

$$\begin{aligned} x(t) &= x_c + a\cos(t) \\ y(t) &= y_c + b\sin(t) \end{aligned} \tag{1.1}$$

where a and b are the length of the major/minor axes corresponding, dependent upon $a > b$ or $a < b$. If we graph this ellipse, starting at $t = 0$, then initially wee have the point $(x_c + a, y_c)$. The bottom most point on the ellipse occurs when $t = \frac{3}{2}\pi$, yielding the point $(x_c, y_c - b)$. So in order to roll an ellipse along

a horizontal line, our line must have the equation

$$y = y_c - b \tag{1.2}$$

To perform actual calculations, we must choose values for the center (x_c, y_c) and lengths of axes, a and b, however it should be apparent throughout the following calculations that nothing depends on these values. We will choose our center to be at $(2, 4)$, with $a = 4$ and $b = 2$. Thus our horizontal line, given in equation (1.2), is $y = 4-2 = 2$. We will have *Maple* graph our initial setup now.

```
> restart:
> with(plots): with(plottools): with(linalg): with(LinearAlgebra):
> xc:= 2; yc:= 4;
```

$$xc := 2$$
$$yc := 4$$

```
> a:= 4; b:= 2;
```

$$a := 4$$
$$b := 2$$

```
> EllipseF:= t->[xc+a*cos(t), yc+b*sin(t)];
```

$$EllipseF := t \to [xc + a\cos(t),\ yc + b\sin(t)]$$

```
> plot1:= plot([xc+a*cos(t), yc+b*sin(t), t=0..2*Pi], scaling = constrained,
color = red, thickness = 3, labels = [x,y]):
> plot2:= plot(2, x = -10..20, color = magenta, thickness = 2):
> plot3:= pointplot([xc, yc], symbol=cross, symbolsize=10, color=black):
> display({plot1, plot2, plot3}, view=[-10..20, 0..17], scaling=constrained);
```

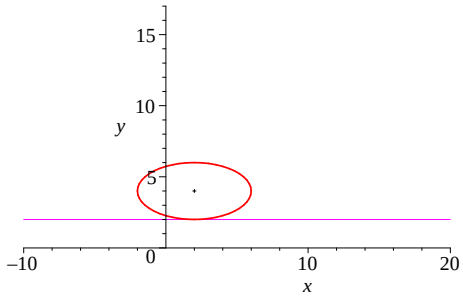

Figure 1.1: The original ellipse and the horizontal line it will roll along

As you can see from Figure 1, the ellipse rests on the horizontal line that it will roll along. So we have now set up our problem.

1.2 The First Step

As a next step in the process, we simply with to roll our ellipse a fixed distance, r, to the right. If we wish to roll the ellipse r units in the x-direction, then we must count out r units on the ellipse, in a counter-clockwise direction, starting at the point on the ellipse which is touching the line.

Recall that the formula for the arclength L of a parametric curve $(x(t), y(t))$ for $t \in (t_0, t_1)$ is given by

$$L = \int_{t_0}^{t_1} \sqrt{(x'(t))^2 + (y'(t))^2}\, dt \tag{1.3}$$

The total arclenth of the ellipse can be found by setting $t_0 = 0$ and $t_1 = 2\pi$, which gives a full revolution.

```
> evalf(int(sqrt(diff(a*cos(t),t)^2 + diff(b*sin(t),t)^2), t=0..2*Pi));
```

19.37689643

For the first step, $t_0 = \frac{3}{2}\pi$, and we need to find the value of t_1 such that

$$r = \int_{\frac{3}{2}}^{t_1} \sqrt{(x'(t))^2 + (y'(t))^2}\, dt$$

Setting $r = 2$, which is reasonably small in comparison to the arclength of the ellipse, means that we need to find the time corresponding to the point (x, y) which is the end of the length 2 arc starting at the point on the ellipse which touches the line. (We are assuming counter-clockwise direction since the ellipse is moving to the right). Setting arclength to 2, gives the equation

$$2 = \int_{\frac{3}{2}}^{t_1} \sqrt{16\sin^2(t) + 4\cos^2(t)}\, dt$$

In order to get the correct solution value for t_1, we need to make sure that $t_1 > \frac{3}{2}\pi$. We will use the *fsolve* command, with the extra option of limiting the range to search for solutions.

```
> fsolve(2 = evalf(int(sqrt(16*sin(t)^2 + 4*cos(t)^2), t=3*Pi/2..t1)), t1, 3*Pi/2
.. 2*Pi);
```

5.229238282

We plot the difference of 2 and the arclength formula, notice the root here corresponds to the t value which give arclength 2.

```
> plot(2 - evalf(int(sqrt(16*sin(t)^2 + 4*cos(t)^2), t=3*Pi/2..t1)), t1=0..2*Pi);
```

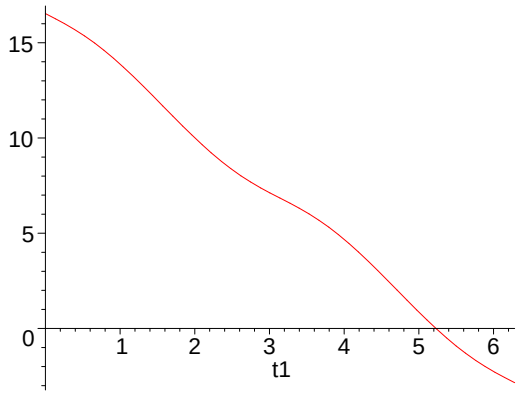

Figure 1.2: Difference of 2 and the arclength of the ellipse

```
> EllipseF(5.229238282);
```

$$[3.976573758, 2.261239222]$$

```
> plot4:= pointplot([EllipseF(3*Pi/2), EllipseF(5.229238282)], symbol = diamond, symbolsize = 20, color = GREEN):
> plot5:= pointplot([4,2], symbol=circle, symbolsize=20, color=black):
> display({plot1, plot2, plot3, plot4, plot5},view=[-3..10,-3..10]);
```

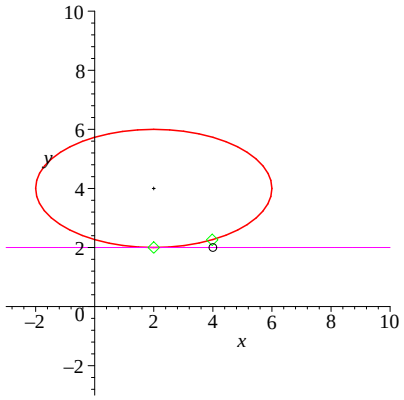

Figure 1.3: Original point on the ellipse which touches the line, and the new point which will touch the line after rolling two units to the right.

> display({plot1, plot2, plot3, plot4, plot5}, view=[1..5, 1..3], scaling = constrained);

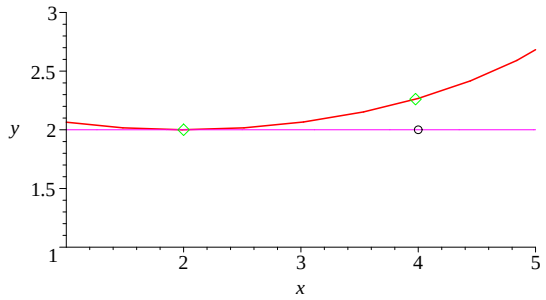

Figure 1.4: Close-up of Figure 1.3 near the desired point of rotation

Now we need to rotate the ellipse. But we are missing one important piece of information, the angle of rotation. We need to construct the pair of vectors which will yield the correct angle of rotation. As a first guess, you may try to use the three points depicted in Figure 1.4 above, but perform the resulting calculations show that this will not work. The next obvious idea is to still use the horizontal line for one vector, but as second vector, use the tangent line at the new point on the ellipse which will be touching the horizontal line after rotation. This also makes use determine where this tangent line intercepts the horizontal line $y = 2$. We illustrate this in the *Maple* code and Figure 1.5 following.

> EllipseF(t);

$$[2 + 4\cos(t), 4 + 2\sin(t)]$$

> M:= diff(EllipseF(t),t)[2]/diff(EllipseF(t),t)[1];

$$M := -\frac{1}{2}\frac{\cos(t)}{\sin(t)}$$

> m:= evalf(subs(t=5.229238282, M));

$$m := 0.2841928836$$

> solve(subs({y=2},y-EllipseF(5.229238282)[2]=m*(x-EllipseF(5.229238282)[1])), x);

$$3.057341655$$

> arrow1:= arrow([3.057341655, 2], [3.976573758, 2.261239222], .02, .1, .1, color = black):

> arrow2:= arrow([3.057341655, 2], [4, 2], .02, .1, .1, color = black):

> plot6:= pointplot([3.057341655,2], symbol = circle, color = BLUE, symbolsize = 20):

> display([plot1, plot2, plot3, plot4, plot5, plot6, arrow1, arrow2], view = [2.9..4.2, 1.8..2.4], scaling = constrained);

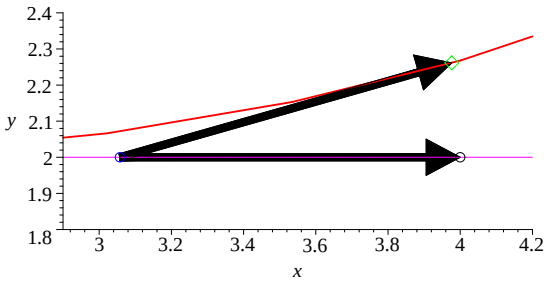

Figure 1.5: The vectors needed to compute the angle of rotation for the first step in the rolling.

> Center:= [3.057341655,2];

$$Center := [3.057341655, 2]$$

> Top:= EllipseF(5.229238282);

$$Top := [3.976573758, 2.261239222]$$

> Bottom:= [4, 2];

$$Bottom := [4, 2]$$

> u:= <Top-Center>;

$$u := \begin{bmatrix} 0.919232103 \\ 0.261239222 \end{bmatrix}$$

> v:= <Bottom-Center>;

$$v := \begin{bmatrix} 0.942658345 \\ 0 \end{bmatrix}$$

> theta_r:= VectorAngle(u,v);

$$theta\_r := 0.2768925149$$

> evalf(%*180/Pi);

$$15.86477248$$

So from above, we have approximately 15.86 degrees to rotate, or 0.27 radians. So now we need to move the ellipse to the origin, rotate, and then finally place

it back so that its center is back at the original center. To do this, we take the ellipse function $F(t)$, and subtract the center $(2, 4)$ from the corresponding components. Next, we rotate using the rotation matrix

$$A_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{1.4}$$

and our formula for the rotated ellipse, $F_\theta(t)$ is now given by

$$F_\theta(t) = A_\theta(F(t) - (2, 4)) + (2, 4). \tag{1.5}$$

```
> EllipseCol:=matrix(2,1,EllipseF(t)-[2,4]);
```

$$EllipseCol := \begin{bmatrix} 4\cos(t) \\ 2\sin(t) \end{bmatrix}$$

```
> Atheta:= theta->matrix(2,2,[cos(theta),-sin(theta),sin(theta),cos(theta)]):
> NewEllipse:= evalm(Atheta(-theta_r)&*EllipseCol);
```

$$NewEllipse := \begin{bmatrix} 3.847638271\cos(t) + 0.5467357072\sin(t) \\ -1.093471414\cos(t) + 1.923819136\sin(t) \end{bmatrix}$$

```
> plot7:= plot([NewEllipse[1,1]+2,NewEllipse[2,1]+4, t=0..2*Pi], color = blue,
thickness = 2):
> display({plot1,plot2,plot3,plot4,plot5,plot6,plot7}, view = [-4..10, -4..10]);
```

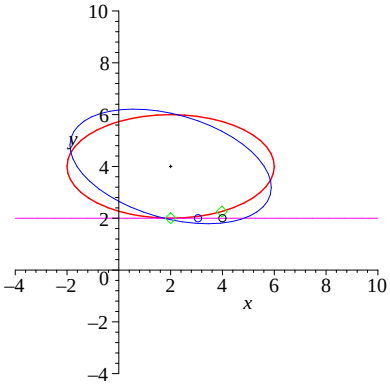

Figure 1.6: Original ellipse and the rotated ellipse, both at center $(2, 4)$

We have successfully rotated the ellipse, now we need to move it into the correct position. This must be a simple shift of some kind, but we need to find the right one, keeping in mind that we wish to automate this process. The one thing we must be sure of, is that the lowest point on the rotated ellipse

(corresponding to $t = 5.229238282$) must move to the point $(4, 2)$ since that is how far to the right we are rolling the ellipse along the horizontal line. Correspondingly, the original center $(2, 4)$ must therefore be shifted the same amount!

```
> LowestPoint:= [evalf(subs({t=5.229238282}, NewEllipse[1,1])), evalf( subs(
{t=5.229238282}, NewEllipse[2,1]))];
```

$$LowestPoint := [1.425963908, -2.212862355]$$

```
> plot8:= pointplot(LowestPoint, color = black, symbol = circle, symbolsize
= 20):
> minimize(NewEllipse[2,1],t=5..5.6);
```

$$-2.212862355$$

```
> ShifT:= Bottom-LowestPoint;
```

$$ShifT := [2.574036092, 4.212862355]$$

```
> NewNewEllipse:= evalm(NewEllipse+ShifT);
```

$$NewNewEllipse := \begin{bmatrix} 3.847638271\cos(t) + .5467357072\sin(t) + 2.574036092 \\ -1.093471414\cos(t) + 1.923819136\sin(t) + 4.212862355 \end{bmatrix}$$

```
> plot9:= plot([NewNewEllipse[1,1], NewNewEllipse[2,1], t=0..2*Pi], color =
blue, thickness=2, scaling=constrained):
> plot10:= pointplot(ShifT, color = black, symbol = cross, symbolsize = 10):
> display({plot1,plot2,plot3,plot4,plot5,plot9,plot10}, view = [-3..10, -3..10]);
```

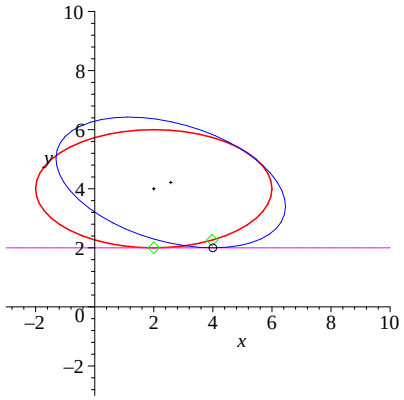

Figure 1.7: Original ellipse and the rotated ellipse, both in the correct positions.

1.3 Automating the Process

Now that we know how an ellipse can be rolled to to right a single fixed distance to the right, we need to determine how to create a process for repeating the process. We will start the process from the previous section over, this time making things more flexible so that the process can be repeated in a *for* loop.

```
> restart:
> with(linalg): with(plots): with(LinearAlgebra):
```

The variables xc and yc will correspond to the center of the ellipse, and a and b are the lengths along the x-axis and y-axis of the ellipse, respectively.

```
> xc:=2; yc:=4;
```

$$xc := 2$$
$$yc := 4$$

```
> a:=2; b:=4;
```

$$a := 4$$
$$b := 2$$

The definition of the ellipse, as a function of t, where $0 \le t \le 2\pi$ is given next.

```
> EllipseF:=[xc+a*cos(t),yc+b*sin(t)];
```

$$EllipseF := [2 + 4\cos(t), 4 + 2\sin(t)]$$

The variable r corresponds to how far to the right we wish to go for each step. The variable *yline* is the y =line that the ellipse will roll along.

```
> r:= 0.5:
> yline:= yc-b;
```

$$yline := 2$$

Next we calculate the total perimeter of the ellipse. This will help is later, when we need to use the *fsolve* command to locate the times that sweep out arclengths of length r given above, which is where we come up with the formula in the definition of the variable S.

```
> totalPerimeter:= evalf(int(sqrt(diff(a*cos(t),t)^2 + diff(b*sin(t),t)^2), t =
0..2*Pi));
```

$$totalPerimeter := 19.37689643$$

The variable *numsteps* tells us how many steps to the right we will go, *Atheta* is the standard rotation matrix, $t0$ is the initial time corresponding to the point on the ellipse that touches the line *yline*. The variable *xval* is where we start the ellipse rolling, in the x-direction.

```
> numrsteps:= 40:
> S:= (totalPerimeter/r)/numrsteps;
```

$$S := 0.9688448215$$

```
> Atheta:= theta->matrix(2,2,[cos(theta),-sin(theta),sin(theta),cos(theta)]):
> t0:= 3*Pi/2:
> xval:= xc:
```

We will want to plot each ellipse, and corresponding center, as we go through the *for* loop. Therefore we initialize a couple of plots.

```
> PlotEllipse[1]:= plot([EllipseF[1], EllipseF[2], t = 0..2*Pi], color = blue,
thickness = 3):
  PlotCenter[1]:= pointplot([xc, yc], symbol = cross, symbolsize = 15, color
= red):
```

We now do the *for* loop which, which automates each step from the previous section. See if you can determine exactly how the following code works, and relate it back to the steps used in Section 1.2.

```
> for k from 1 to numrsteps do
    ts:= fsolve(r = evalf(int(sqrt(diff(EllipseF[1],t)^2 + diff(EllipseF[2],t)^2),
      t=t0..t1)), t1,t0..t0+3*S):
    M:= diff(EllipseF[2], t)/diff(EllipseF[1], t):
    m:= evalf(subs(t=ts, M));
    xint:= solve(subs({y=yline}, y-subs({t=ts},EllipseF[2])=m*(x- subs(t=ts,
      EllipseF[1]))), x):
    Cpt:= [xint,yline]:
    Tpt:= [evalf(subs(t=ts,EllipseF[1])), evalf(subs(t=ts,EllipseF[2]))]:
    Bpt:= [xval+k*r, yline]:
    u:= <evalf(Tpt-Cpt)>:
    v:= <evalf(Bpt-Cpt)>:
    theta_r:= evalf(VectorAngle(u,v)):
    EllipseCol:= evalf(EllipseF-[xc,yc]):
    NewEllipse:= evalm(Atheta(-theta_r)&*EllipseCol):
    Lpt:= [evalf(subs(t=ts,NewEllipse[1])), evalf(subs(t=ts, NewEllipse[2]))]:
    ShifT:= evalf([Bpt[1]-Lpt[1], Bpt[2]-Lpt[2]]):
```

```
  TempFunc:= [evalm(NewEllipse+ShifT)[1], evalm(NewEllipse+ShifT)[2]]:
  EllipseF:= TempFunc:
  xc:= ShifT[1]:
  yc:= ShifT[2]:
  PlotEllipse[k+1]:= plot([EllipseF[1],EllipseF[2],t=0..2*Pi], color = blue,
    thickness = 3):
  PlotCenter[k+1]:= pointplot([xc,yc], symbol = cross, symbolsize = 15,
    color = red):
  print([evalf(t0), evalf(ts), evalf(m), xint, theta_r]);
  t0:=ts:
end do:
```

```
[4.712388981, 4.837634224, 0.06295213005, 2.250818442, 0.06286916815]
[4.837634224, 4.964377945, 0.06524991473, 2.752337634, 0.06515755071]
[4.964377945, 5.094260323, 0.07024557186, 3.253993975, 0.07013037181]
[5.094260323, 5.229238281, 0.07889852891, 3.755899408, 0.07873541969]
[5.229238281, 5.371839344, 0.09315749557, 4.258205308, 0.09288940404]
  [5.371839344, 5.525564365, 0.1169956373, 4.761116197, 0.1164661717]
  [5.525564365, 5.695516530, 0.1589181977, 5.264841596, 0.1576002841]
  [5.695516530, 5.889027416, 0.2374022491, 5.769127160, 0.2330872702]
  [5.889027416, 6.113472938, 0.3806155108, 6.270699972, 0.3636847478]
  [6.113472938, 6.360840027, 0.5264205469, 6.759057056, 0.4845599565]
  [6.360840027, 6.596574059, 0.4473881505, 7.238960154, 0.4206797897]
  [6.596574059, 6.801150074, 0.2827341251, 7.732725761, 0.2755422530]
  [6.801150074, 6.979000636, 0.1827302672, 8.234910919, 0.1807362574]
  [6.979000636, 7.138131068, 0.1299721980, 8.738278564, 0.1292476649]
  [7.138131068, 7.284443626, 0.1006979579, 9.241201103, 0.1003596554]
[7.284443626, 7.421956334, 0.08346104147, 9.743577472, 0.08326805914]
[7.421956334, 7.553505394, 0.07300086043, 10.24554568, 0.07287159184]
[7.553505394, 7.681226798, 0.06677931948, 10.74724501, 0.06668031490]
[7.681226798, 7.806856500, 0.06353635545, 11.24878696, 0.06345106277]
[7.806856500, 7.931928746, 0.06269043691, 11.75026255, 0.06260850623]
[7.931928746, 8.057926053, 0.06409838242, 12.25175412, 0.06401081046]
[8.057926053, 8.186417244, 0.06800045527, 12.75334713, 0.06789592349]
[8.186417244, 8.319215528, 0.07511350205, 13.25514343, 0.07497271764]
[8.319215528, 8.458595981, 0.08693581881, 13.75727742, 0.08671779473]
  [8.458595981, 8.607632126, 0.1064909067, 14.25993315, 0.1060910764]
  [8.607632126, 8.770734819, 0.1401247346, 14.76333529, 0.1392182749]
```

[8.770734819, 8.954371422, 0.2017240386, 15.26752838, 0.1990527406]
[8.954371422, 9.166689926, 0.3181753113, 15.77094742, 0.3080468784]
[9.166689926, 9.408443977, 0.4868817082, 16.26561855, 0.4530980124]
[9.408443977, 9.652710266, 0.5042403084, 16.74558633, 0.4670341024]
[9.652710266, 9.869234493, 0.3388261544, 17.23342977, 0.3266859207]
[9.869234493, 10.05615447, 0.2131173091, 17.73374068, 0.2099759606]
[10.05615447, 10.22152186, 0.1461548899, 18.23703360, 0.1451273440]
[10.22152186, 10.37210920, 0.1098907450, 18.74016874, 0.1094515809]
[10.37210920, 10.51255477, 0.08896094943, 19.24273800, 0.08872737794]
[10.51255477, 10.64607099, 0.07634623448, 19.74484234, 0.07619841721]
[10.64607099, 10.77501472, 0.06872534658, 20.24662846, 0.06861744890]
[10.77501472, 10.90124534, 0.06445684485, 20.74821819, 0.06436780345]
[10.90124534, 11.02635382, 0.06274518131, 21.24970894, 0.06266303789]
[11.02635382, 11.15182601, 0.06329655010, 21.75118488, 0.06321222467]

```
> Yplot:=plot(yline,x=-3..30,color=BLACK, thickness=2):
> for j from 1 to numrsteps+1 do
    PlotTog[j]:=display([PlotEllipse[j],PlotCenter[j], Yplot]):
  end do:
> display([seq(PlotTog[l],l=1..numrsteps+1)], insequence = true, scaling =
constrained, axes = none);
```

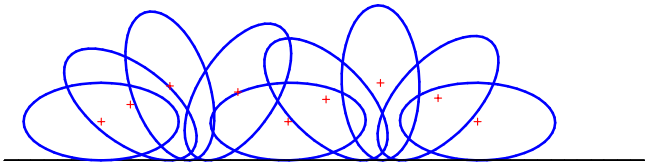

Figure 1.8: Nine frames in the animation of rolling the ellipse

Figure 1.8 is the culmination of all of our work thus far. Unfortunately, we cannot display the actual animation here, so nine frames are show at the same time. The lighter ellipse on the left is the first in the series. Also note that each subsequent ellipses rests on the horizontal line, even after being rotated. So it appears that we have successfully rolled an ellipse along a horizontal line!

> display(seq(PlotCenter[j], j = 1..numrsteps+1), view = [0..25, 0..10], scaling = constrained);

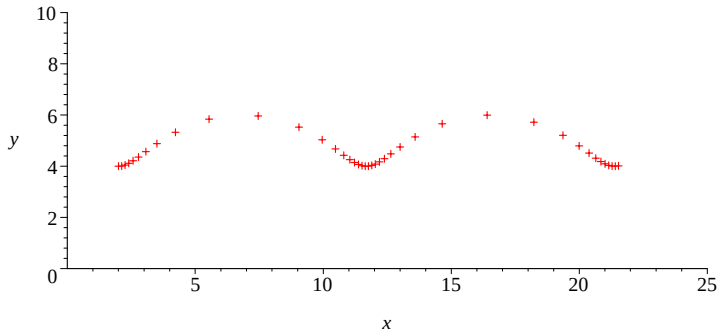

Figure 1.9: The centers of each of the ellipses throughout the rotation

Showing all 41 ellipses at the same time would not help us view the process at all. However, in Figure 1.9, the center of each ellipse is displayed. Note that the distance between centers is significantly less when the curvature of the portion of the ellipse resting on the line is larger. This corresponds to the portions of the ellipse closest to the minor axes.

1.4 Further Questions to Consider

We end with a discussion (and leave you with questions) on how to generalize this process. Before we do this, we must realize what restrictions were used in all the work done previous. The most important assumption that was made, was that our curve that we rolled the ellipse along was actually a horizontal line.

1. How do things change if we decide not to use a horizontal line, but a line with nonzero slope instead?

If you can determine the answer to this question, then the following should be answerable as well. Remember, that if we were rolling an ellipse a distance r to the right along a horizontal line, where the ellipse touched the line at the point (x_p, y_p), we then knew that the rotated ellipse would be touching the horizontal line at the point $(x_p + r, y_p)$. This is clearly not the case for a line with nonzero slope! Furthermore, we also use the fact that the line was horizontal to help compute the angle of rotation needed. This must change as well! If you can figure out how to adapt the previous work to allow for lines on nonzero slope, then the following questions may also be answerable.

2. What must be modified to roll an ellipse along a curve given in the form $y = f(x)$?

3. What happens if you wish to roll an ellipse along a parametric curve of the form $(x, y) = (x(s), y(s))$? Is this really a generalization of question 1 or is more mathematics required?