2019-11-04 : crashing the glider!

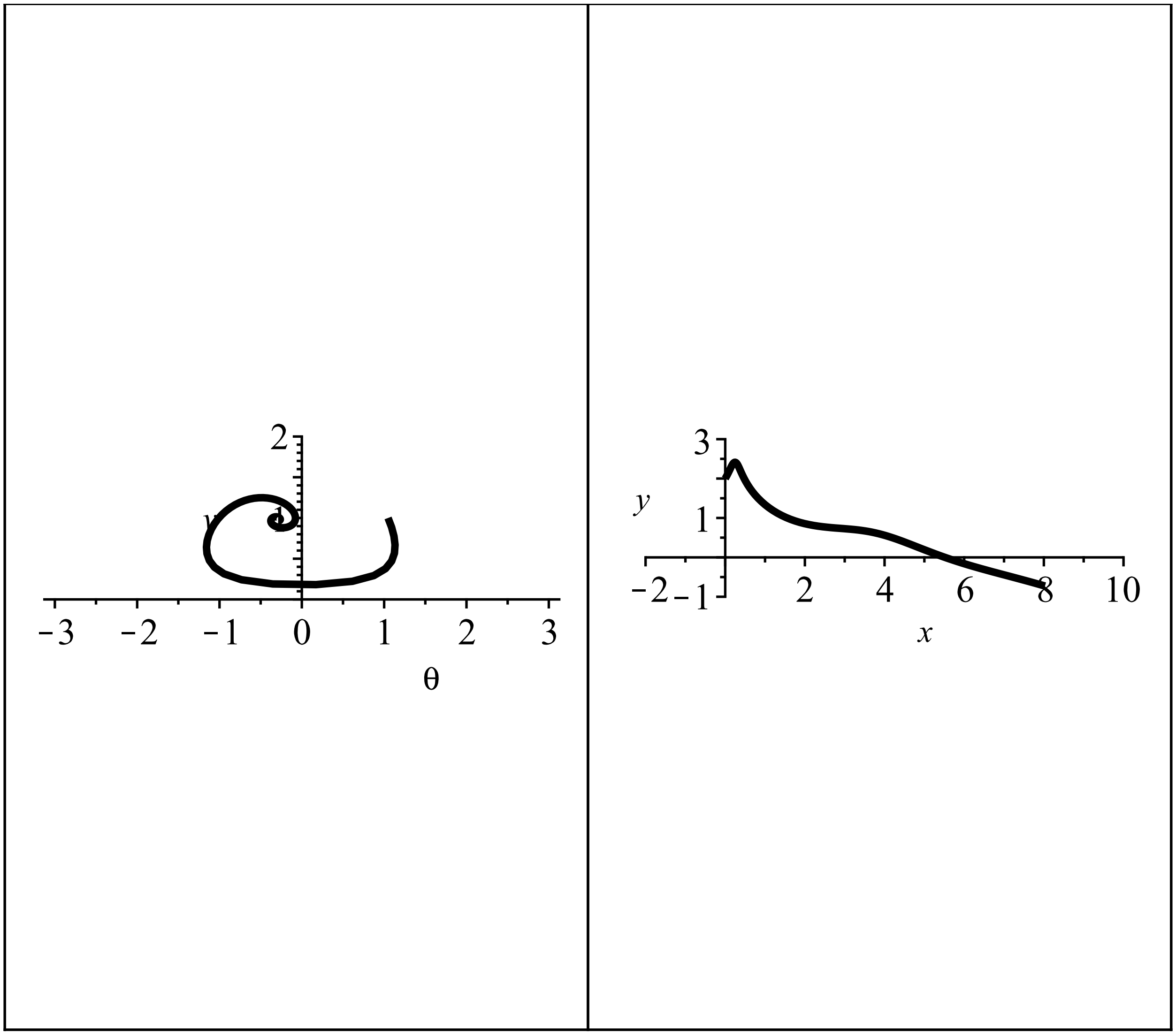
> $with(DEtools)$:

> $xphug(R) := \left\{ diff(\mathrm{theta}(t), t) = v(t) - \dfrac{\cos(\mathrm{theta}(t))}{v(t)}, diff(v(t), t) = -\sin(\mathrm{theta}(t)) - R \cdot v(t)^2, diff(x(t), t) = v(t) \cdot \cos(\mathrm{theta}(t)), diff(y(t), t) = v(t) \cdot \sin(\mathrm{theta}(t)) \right\};$

$$xphug := R \mapsto \left\{ \frac{\mathrm{d}}{\mathrm{d}t}\, \theta(t) = v(t) - \frac{\cos(\theta(t))}{v(t)}, \frac{\mathrm{d}}{\mathrm{d}t}\, v(t) = -\sin(\theta(t)) - R\, v(t)^2, \frac{\mathrm{d}}{\mathrm{d}t}\, x(t) = v(t) \cos(\theta(t)), \frac{\mathrm{d}}{\mathrm{d}t}\, y(t) = v(t) \sin(\theta(t)) \right\} \tag{1}$$

Let's confirm that the equations are entered correctly.

> $plots[display]\Big(\Big\langle$
$DEplot\Big(xphug(.3), [\mathrm{theta}(t), v(t), x(t), y(t)], t = 0..10, \left[\left[\mathrm{theta}(0) = \dfrac{\mathrm{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2\right]\right], \mathrm{theta} = -\mathrm{Pi}..\mathrm{Pi}, v = 0..2, x = -2..10, y = -1..3, \quad obsrange = false, linecolor = black, stepsize = .1, scene = [\mathrm{theta}, v], scaling = constrained\Big) \Big|$
$DEplot\Big(xphug(.3), [\mathrm{theta}(t), v(t), x(t), y(t)], t = 0..10, \left[\left[\mathrm{theta}(0) = \dfrac{\mathrm{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2\right]\right], \mathrm{theta} = -\mathrm{Pi}..\mathrm{Pi}, v = 0..2, x = -2..10, y = -1..3, \quad obsrange = false, linecolor = black, stepsize = .1, scene = [x, y], scaling = constrained\Big) \Big\rangle\Big)$



Yup, looks as we expect. Note that the glider "flies" underground... that is, when y<0.

Now let's change the equations so it doesn't fly when y<0.

$$> crasher(R) := \Big\{ diff(\mathrm{theta}(t), t) = piecewise\Big(y(t) > 0, v(t) - \frac{\cos(\mathrm{theta}(t))}{v(t)}, 0\Big),$$
$$diff(v(t), t) = piecewise\big(y(t) > 0, -\sin(\mathrm{theta}(t)) - R \cdot v(t)^2, 0\big),$$
$$diff(x(t), t) = piecewise(y(t) > 0, v(t) \cdot \cos(\mathrm{theta}(t)), 0),$$
$$diff(y(t), t) = piecewise(y(t) > 0, v(t) \cdot \sin(\mathrm{theta}(t)), 0) \Big\} :$$

$$> plots[display]\Big(\Big\langle$$
$$DEplot\Big(crasher(.3), [\mathrm{theta}(t), v(t), x(t), y(t)], t = 0..10,$$
$$\Big[\Big[\mathrm{theta}(0) = \frac{\mathrm{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2\Big]\Big],$$
$$\mathrm{theta} = -\mathrm{Pi}..\mathrm{Pi}, v = 0..2, x = -2..10, y = -1..3, \quad obsrange = false, linecolor = black, stepsize$$

$= .1,$

$\quad scene = [\text{theta}, v], scaling = constrained\Big)$

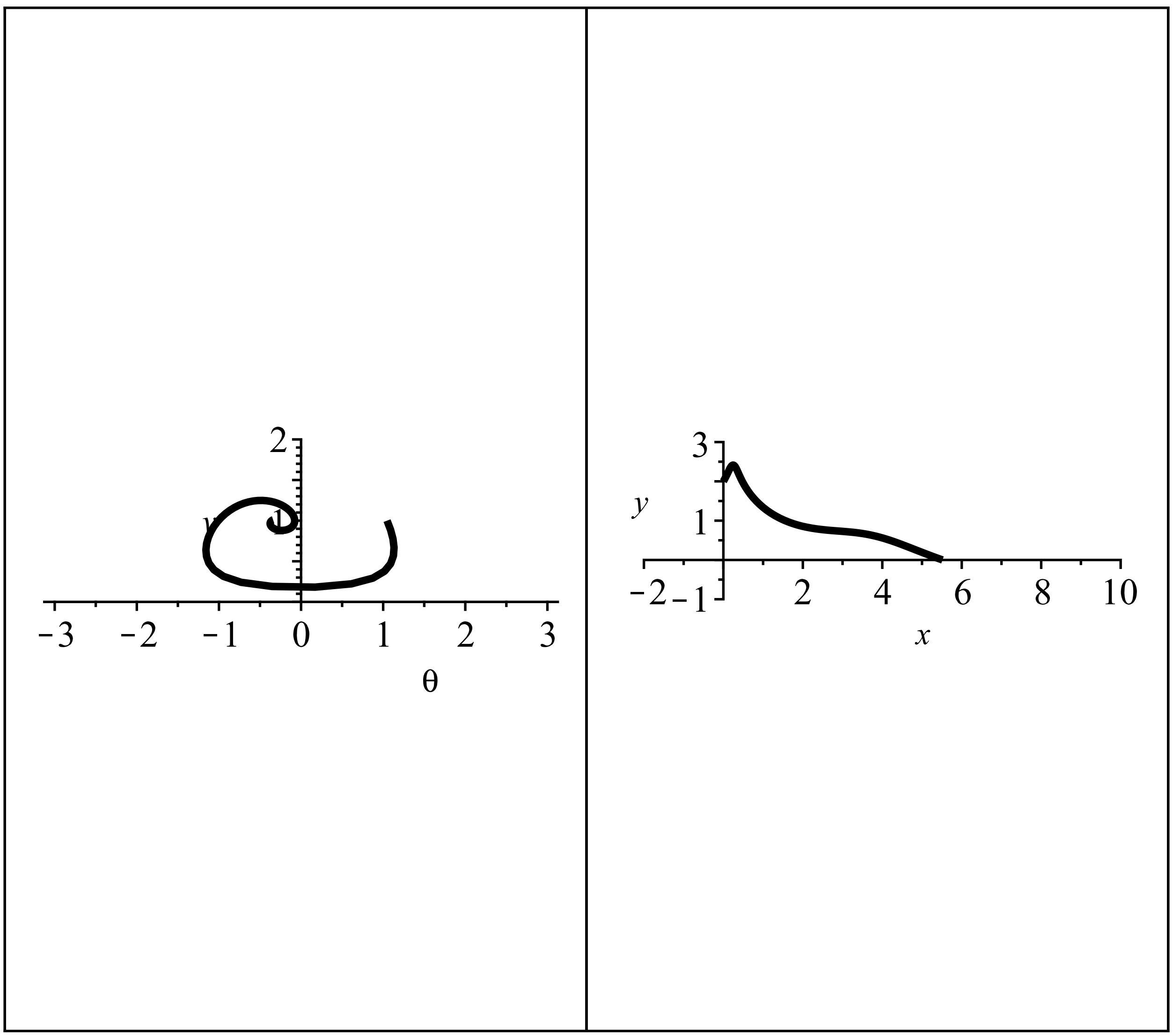
$DEplot\Big(\; crasher(.3),\ [\text{theta}(t), v(t), x(t), y(t)], t = 0\,..10,$

$\quad \Big[\Big[\text{theta}(0) = \frac{\text{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2\Big]\Big],$

$\quad \text{theta} = -\text{Pi}\,..\text{Pi}, v = 0\,..2, x = -2\,..10, y = -1\,..3, \quad obsrange = false, linecolor = black, stepsize$

$= .1,$

$\quad scene = [x, y], scaling = constrained\Big)\Big\rangle\Big)$



That looks like it worked.

Our goal for today is to write a function that, given an input angle theta(0), tells us how far the glider went before hitting the ground (assuming a fixed input velocity v(0), starting height (y(0)), and drag constant (R)

let's use dsolve(..., numeric)

> $solPi3 := dsolve\left(\left[op(crasher(.3)), \text{theta}(0) = \frac{\text{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2\right], numeric\right)$

$$solPi3 := \textbf{proc}(x\_rkf45) \;\ldots\; \textbf{end proc} \tag{2}$$

> $solPi3(2)$

$$\left[t = 2., \theta(t) = -1.06295629847427, v(t) = 0.926127351480627, x(t) = 0.515162162276087, y(t) = 1.92503648205329\right] \tag{3}$$

> $solPi3(10)$

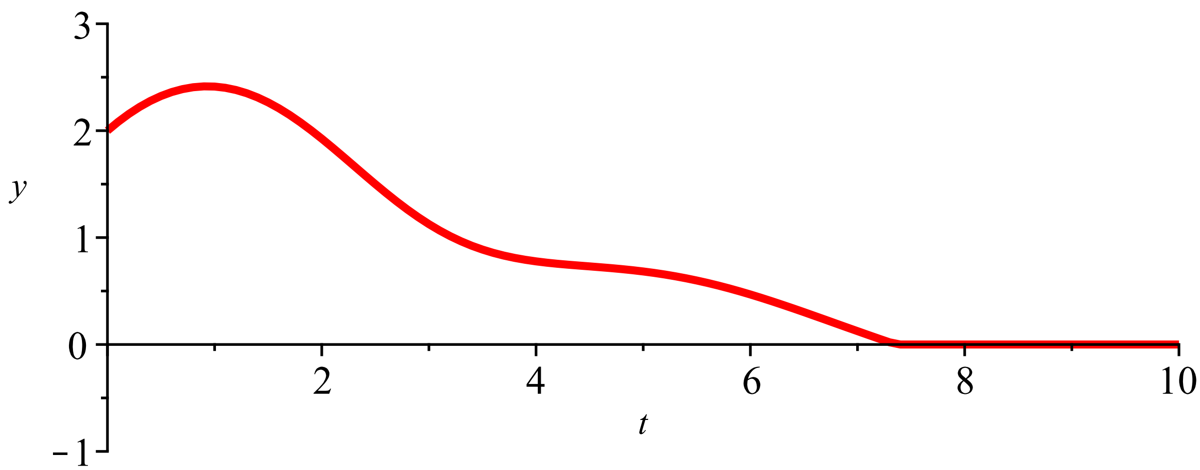
$$\left[t = 10., \theta(t) = -0.340292597271623, v(t) = 1.01025112998594, x(t) = 5.51606970068429, y(t) = -2.33519505139458\ 10^{-7}\right] \tag{4}$$

> $solPi3(8)$

$$\left[t = 8., \theta(t) = -0.340292597271623, v(t) = 1.01025112998594, x(t) = 5.51606970068429, y(t) = -2.33519505139458\ 10^{-7}\right] \tag{5}$$

When did it hit the ground? I can just see that it hit sometime between t=2 and t=8....

> $DEplot\Big(crasher(.3), [\text{theta}(t), v(t), x(t), y(t)], t = 0..10,$
$\left[\left[\text{theta}(0) = \frac{\text{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2\right]\right],$
$\text{theta} = -\text{Pi}..\text{Pi}, v = 0..2, x = -2..10, y = -1..3, \quad obsrange = false, linecolor = red, stepsize$
$= .1,$
$scene = [t, y], scaling = constrained\Big)$



Just in case we want to know WHEN it crashed, we can add a clock that stops when the plane hits the ground. That is, we add a variable which starts at 0 when t=0, and has derivative 1 with respect to t if y>0, but derivative 0 when y<0.

$$> \ crasher(R) := \Big\{ diff(\mathrm{theta}(t), t) = piecewise\Big(y(t) > 0, v(t) - \frac{\cos(\mathrm{theta}(t))}{v(t)}, 0\Big),$$
$$diff(v(t), t) = piecewise\big(y(t) > 0, -\sin(\mathrm{theta}(t)) - R \cdot v(t)^2, 0\big),$$
$$diff(x(t), t) = piecewise(y(t) > 0, v(t) \cdot \cos(\mathrm{theta}(t)), 0),$$
$$diff(y(t), t) = piecewise(y(t) > 0, v(t) \cdot \sin(\mathrm{theta}(t)), 0),$$
$$diff(clock(t), t) = piecewise(y(t) > 0, 1, 0)\Big\}$$

$$crasher := R \mapsto \left\{ \frac{\mathrm{d}}{\mathrm{d}t}\, clock(t) = \begin{cases} 1 & 0 < y(t) \\ 0 & otherwise \end{cases}, \frac{\mathrm{d}}{\mathrm{d}t}\, \theta(t) \right. \tag{6}$$
$$= \begin{cases} v(t) - \dfrac{\cos(\theta(t))}{v(t)} & 0 < y(t) \\ 0 & otherwise \end{cases}, \frac{\mathrm{d}}{\mathrm{d}t}\, v(t)$$
$$= \begin{cases} -\sin(\theta(t)) - R\, v(t)^2 & 0 < y(t) \\ 0 & otherwise \end{cases}, \frac{\mathrm{d}}{\mathrm{d}t}\, x(t) = \begin{cases} v(t)\cos(\theta(t)) & 0 < y(t) \\ 0 & otherwise \end{cases}, \frac{\mathrm{d}}{\mathrm{d}t}$$

$$y(t) = \begin{cases} v(t) \sin(\theta(t)) & 0 < y(t) \\ 0 & otherwise \end{cases}$$

> $solPi3 := dsolve\left(\left[op(crasher(.3)), \mathrm{theta}(0) = \frac{\mathrm{Pi}}{3}, v(0) = 1, x(0) = 0, y(0) = 2, clock(0) = 0\right], numeric\right)$

$$solPi3 := \mathbf{proc}(x\_rkf45) \ \ldots \ \mathbf{end\ proc} \tag{7}$$

> $solPi3(8)$

$$\left[t = 8., clock(t) = 7.35931992790298, \theta(t) = -0.340292614839344, v(t) = 1.01025112201649, x(t) = 5.51606955357705, y(t) = -1.72119202716075\ 10^{-7}\right] \tag{8}$$

> $solPi3(20)$

$$\left[t = 20., clock(t) = 7.35931992790298, \theta(t) = -0.340292614839344, v(t) = 1.01025112201649, x(t) = 5.51606955357705, y(t) = -1.72119202716075\ 10^{-7}\right] \tag{9}$$

> $solPi3(6)$

$$\left[t = 6., clock(t) = 6., \theta(t) = -0.339277815709023, v(t) = 0.907897114988835, x(t) = 4.28806699038502, y(t) = 0.467817414868425\right] \tag{10}$$

So we can see that clock(t)=t as long as y>0, but as soon as y<0, the clock stops.

Recall, we want a function that given initial angle, gives x when it hits.

First step, write a function that solves de with given initial theta.

To write θ_0, type theta__0

Here, R is an optional second parameter that has value .3 if not specified, but allows us to change R if we want.

```
> solfunc :=proc(θ_0, R := .3)
    return(
    dsolve( [op(crasher(R)), theta(0) = θ_0, v(0) = 1, x(0) = 0, y(0) = 2, clock(0) = 0], numeric)
    );
  end:
```

> $solfunc\left(\frac{\mathrm{Pi}}{3}\right)(20)$ # *here, R=.3*

$$\left[t = 20., clock(t) = 7.35931992790298, \theta(t) = -0.340292614839344, v(t) = 1.01025112201649, x(t) = 5.51606955357705, y(t) = -1.72119202716075\ 10^{-7}\right] \tag{11}$$

> $solfunc\left(\frac{\mathrm{Pi}}{3}, .1\right)(20)$ # *here R=.1*

$$\left[t = 20., clock(t) = 19.0248272973520, \theta(t) = -0.106300008391200, v(t) = 0.949596788719964, x(t) = 17.1392773541023, y(t) = -1.14985759463191\ 10^{-8}\right] \tag{12}$$

> $solfunc(0)(20)$

$$\left[t = 20., clock(t) = 7.52230198342977, \theta(t) = -0.299702064083077, v(t) = 0.984795466218774, x(t) = 6.90830624735042, y(t) = -2.03465257224724\ 10^{-7}\right] \tag{13}$$

Then, we can pick out the value of x when it crashes using eval, as in

> $eval(x(t), \mathbf{(13)})$

$$6.9083062474 \tag{14}$$

Or, as a function:

> $xcrash := \text{theta} \rightarrow eval(x(t), solfunc(\text{theta})(20))$

$$xcrash := \theta \mapsto eval\big(x(t), solfunc(\theta)(20)\big) \tag{15}$$

> $xcrash\left(\frac{\text{Pi}}{8}\right)$

$$6.7652515213 \tag{16}$$

The following doesn't work as we would expect:

> $plot(xcrash(angle), angle = -\text{Pi}..\text{Pi})$

Warning, The use of global variables in numerical ODE problems is deprecated, and will be removed in a future release. Use the 'parameters' argument instead (see ?dsolve,numeric,parameters)

Error, (in unknown) parameter 'angle' must be assigned a numeric value before obtaining a solution

What is going on? Why does it hate me? (I know, I'm doing this for effect).

Let's try something simpler. **A digression follows.**
Let's try the same thing with a function which returns x if x<1, and x^2 if x>= 1.

> $xquad := \mathbf{proc}(x)$
    $\mathbf{if}\ (x > 1)\ \mathbf{then}\ \ \mathbf{return}(x^2);$
    $\mathbf{else}\ \ \mathbf{return}(x);$
    $\mathbf{fi};$
   $\mathbf{end}:$

This suffers from the same problem, which is not at all obvious (even though the error message is different, it is the same problem)

> $plot(xquad(x), x = 0..2)$

Error, (in xquad) cannot determine if this expression is true or false: 1 < x

> $xquad(.5), xquad(2)$

$$0.5, 4 \tag{17}$$

Note that the function works when called with numeric arguments, but gives the error (which makes sense) when called with a symbolic one:

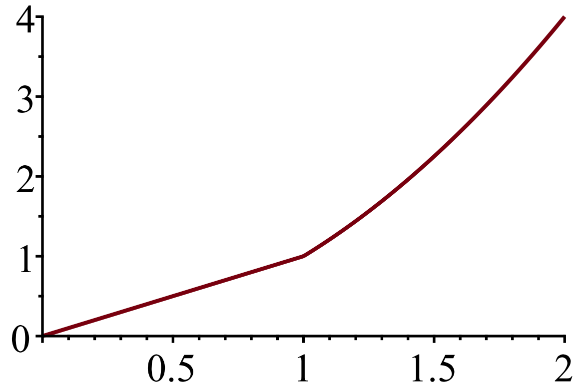
> $xquad(whaa)$

Error, (in xquad) cannot determine if this expression is true or false: 1 < whaa

What is happening is that when plot is called, maple tries to evaluate xquad(x) first (so that it can see if it is able to do some simplification of the argument to plot), then it calls it with values of x between 0 and 2 to generate the plot.

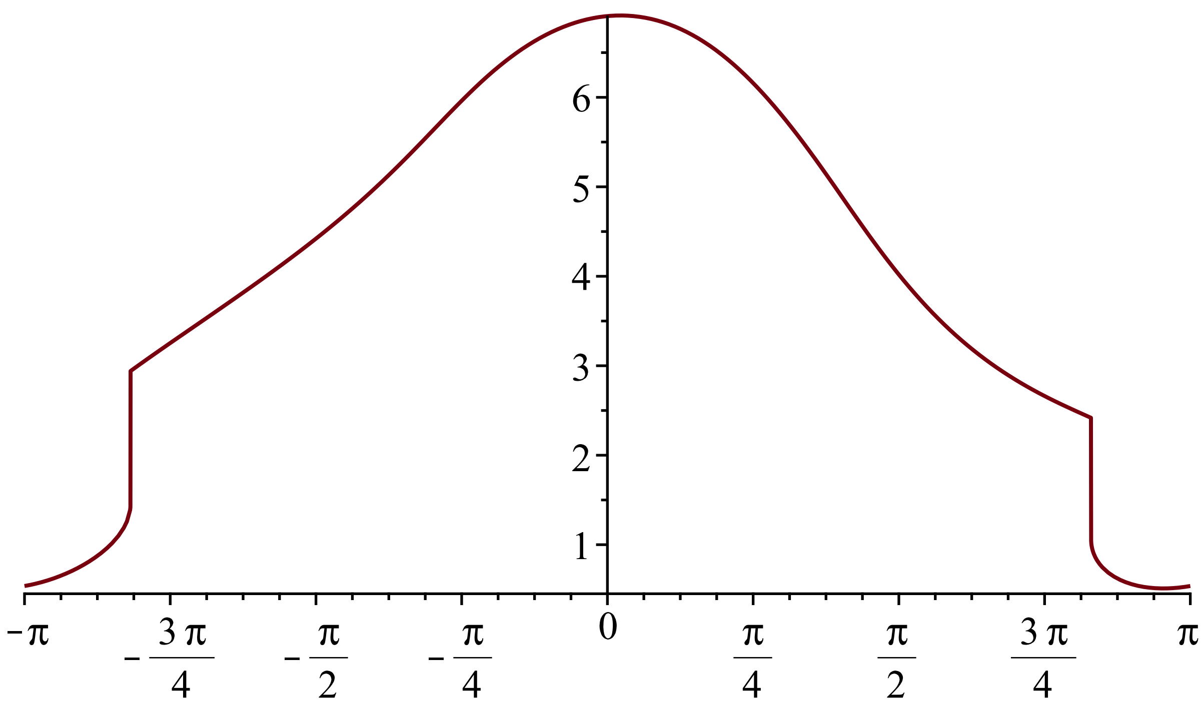
One way around this issue is to just call plot WITHOUT specifying a variable.

> $plot(xquad, 0..2)$



Yup, that works.   Let's try it with xcrash

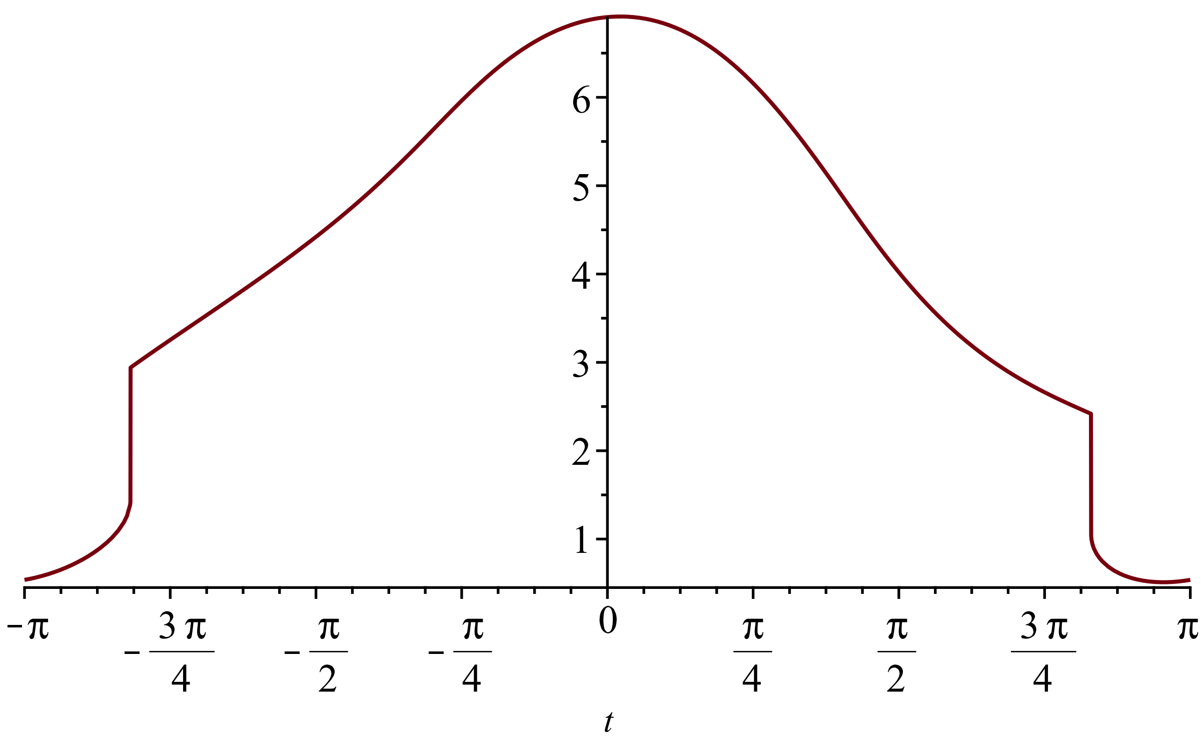
> $plot(xcrash, -\mathrm{Pi}..\mathrm{Pi})$



Hey, cool! it worked.

But let's try another way... maybe I really want to say the input variable. If I put the call to the function in single quotes, this tells maple not to try to evaluate it right away.

> $plot('xcrash'(t), t = -\mathrm{Pi}..\mathrm{Pi})$



However, this is kind of a pain in the butt, to remember to put it in quotes all the time.
We can be clever and check if the passed in value is actually a number. If it is, go ahead and evaluate the function. If it isn't, then just return the name of the function inside quotes. That way, if we forget the quotes, it still works.

We can check whether something is numeric or not using the maple command type (similar to whattype):

```
> whattype(x)
```

symbol **(18)**

```
> whattype(32.6), whattype(6)
```

float, integer **(19)**

so x is a symbol, and 32.6 is a float, and 6 is an integer.

```
> type(x, numeric), type(32.6, numeric), type(6, numeric), type(Pi, numeric)
```

false, true, true, false **(20)**

Wait, why isn't Pi numeric? Cuz it isn't a number, it is a symbol that evaluates to a number if we want to approximate it.

```
> whattype(evalf(Pi))
```

float **(21)**

OK, so we can ask if evalf() of the argument is of type numeric. If so, go ahead and plug it in. If not, just return a quoted version of the function.

```
> xcrash := proc(ang)
    local crashvals;
    if (type(evalf(ang), numeric)) then
      crashvals := solfunc(ang)(20);
      return(eval(x(t), crashvals));
    fi;
    return('xcrash'(ang));
```

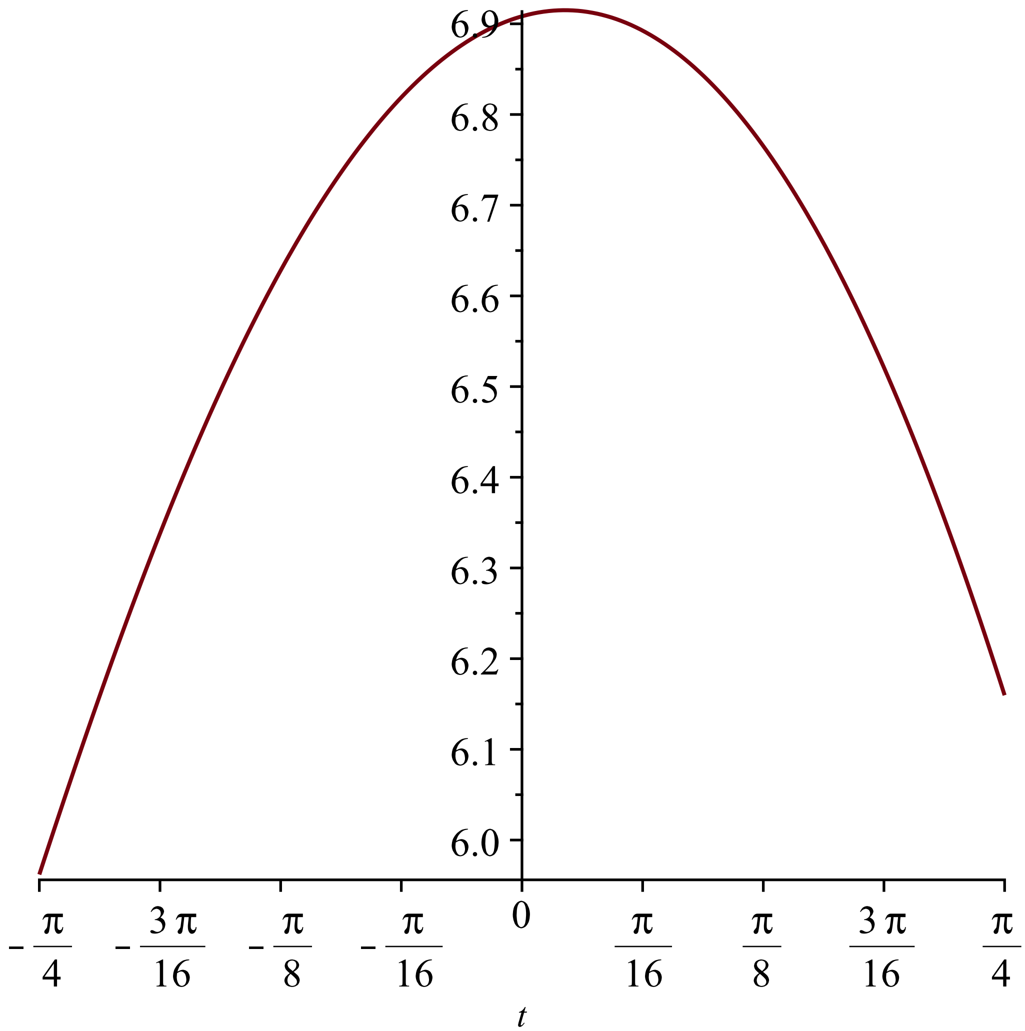
end:

Does it behave as we expect?

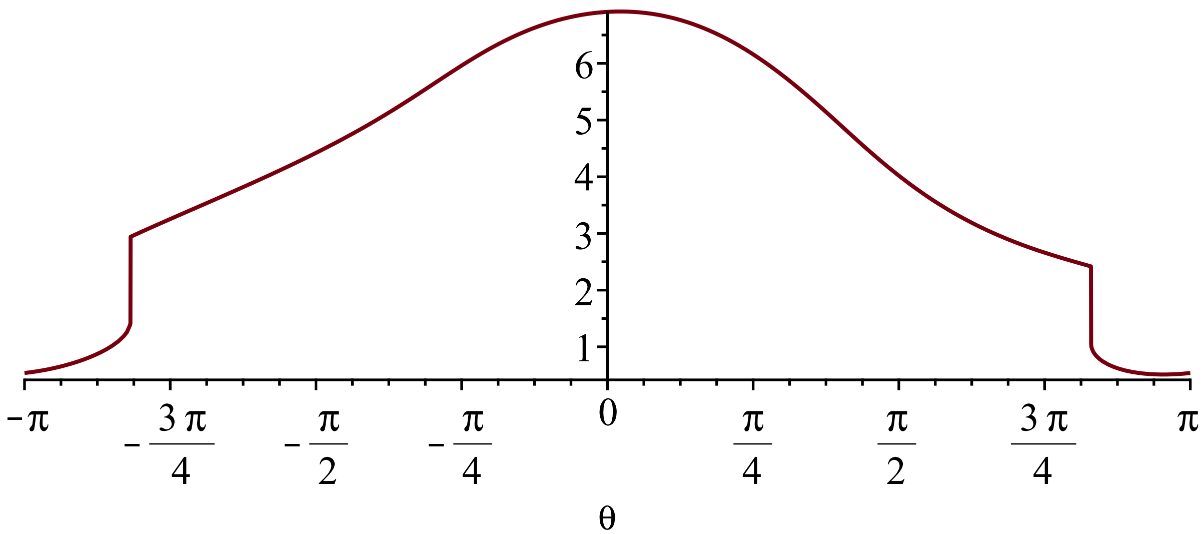
> $xcrash(.1), xcrash(stuff), xcrash(\mathrm{Pi})$

$$6.9137506885, xcrash(stuff), 0.5381780817 \quad (22)$$

> $plot\left(xcrash(t), t=-\frac{\mathrm{Pi}}{4} .. \frac{\mathrm{Pi}}{4}\right)$



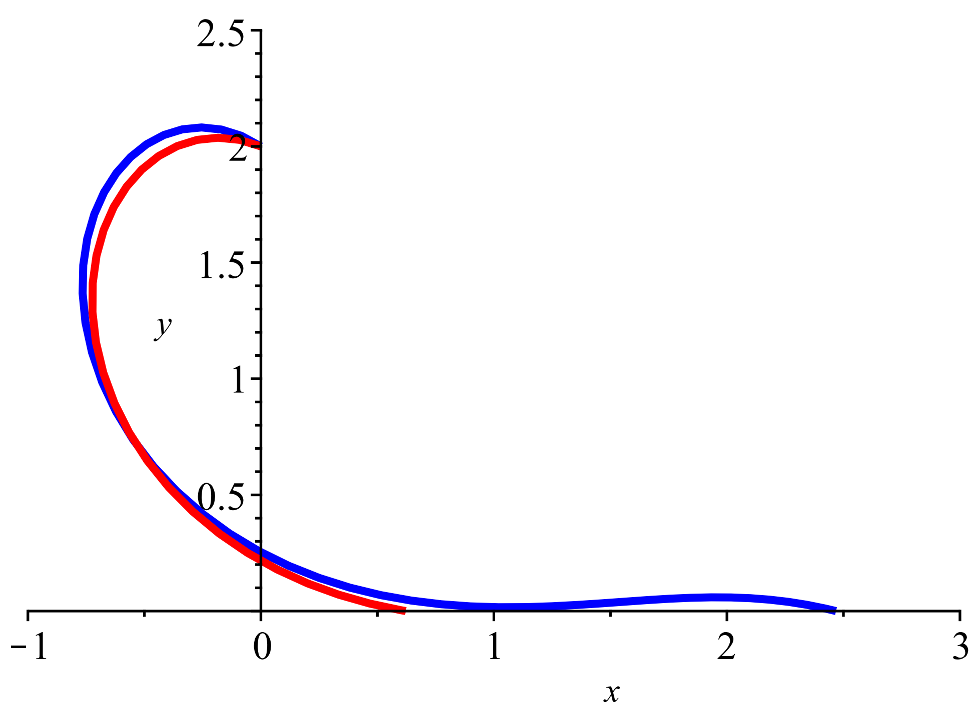
> $plot(xcrash(\mathrm{theta}), \mathrm{theta}=-\mathrm{Pi} .. \mathrm{Pi})$



>

What is this weirdness just above 3Pi/4 ?
Let's take a look.

> $DEplot\Big(crasher(.3), [\mathrm{theta}(t), v(t), x(t), y(t), clock(t)], t = 0..10,$
$\Big[\Big[\mathrm{theta}(0) = \frac{13\cdot \mathrm{Pi}}{16}, v(0) = 1, x(0) = 0, y(0) = 2, clock(0) = 0\Big],$
$\Big[\mathrm{theta}(0) = \frac{14\cdot \mathrm{Pi}}{16}, v(0) = 1, x(0) = 0, y(0) = 2, clock(0) = 0\Big]\Big],$
$x = -1..3, y = 0..2.5, \quad obsrange = false, linecolor = [blue, red], stepsize = .1,$
$scene = [x, y], scaling = constrained\Big)$



So, for angles too close to Pi, the glider crashes before it fully flips over, but those further away just miss crashing and manage to go further. This gives a discontinuity in the distance travelled.

But wait.... We forgot about the ability to vary the drag parameter R (which was written into solfunc) Just as a reminder:

> $solfunc\left(\frac{14\,\mathrm{Pi}}{6}\right)(20)$ # *uses R=.3*

$[t = 20.0000000000, clock(t) = 7.3593192736, \theta(t) = 5.9428928186, v(t) = 1.0102512759, x(t) = 5.5160687635, y(t) = -3.0360274457\ 10^{-7}]$ **(23)**

> $solfunc\left(\frac{14\,\mathrm{Pi}}{6}, .3\right)(20)$

$[t = 20.0000000000, clock(t) = 7.3593192736, \theta(t) = 5.9428928186, v(t) = 1.0102512759, x(t) = 5.5160687635, y(t) = -3.0360274457\ 10^{-7}]$ **(24)**

> $solfunc\left(\frac{14\,\mathrm{Pi}}{6}, .1\right)(20)$ # *R=.1*

$[t = 20.0000000000, clock(t) = 19.0248344358, \theta(t) = 6.1768837469, v(t) = 0.9495957873, x(t) = 17.1392817040, y(t) = -7.1633861937\ 10^{-8}]$ **(25)**

We want to add a parameter R to xcrash.   I forgot about it before, so here it is.

```
> xcrash := proc(ang, R := .3)
    local crashvals;
    if (type(evalf(ang), numeric)) then
      crashvals := solfunc(ang, R)(20);
      return(eval(x(t), crashvals));
    fi;
    return('xcrash'(ang, R));
  end:
```

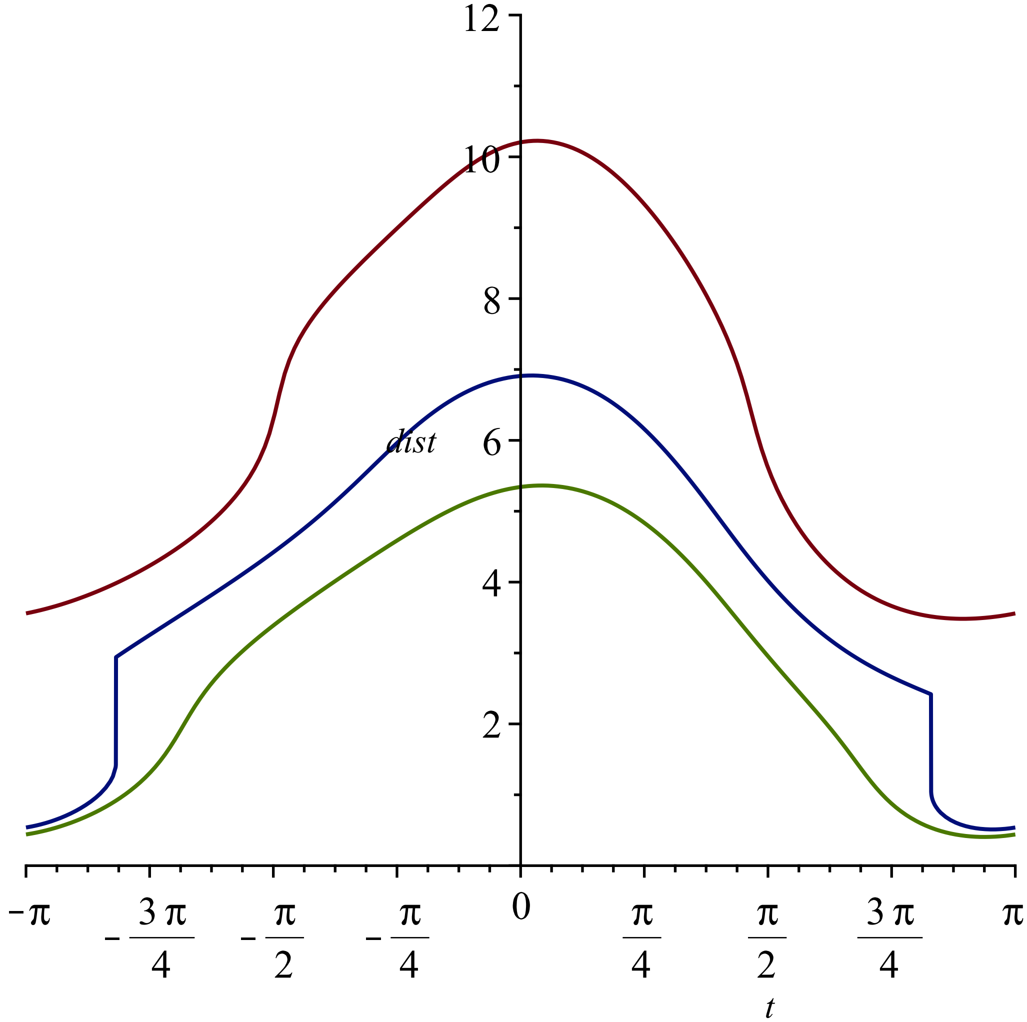
> $xcrash\left(\frac{14\,\mathrm{Pi}}{6}\right)$ # *R=.3*

5.5160687635 **(26)**

> $xcrash\left(\frac{14\,\mathrm{Pi}}{6}, .1\right)$

17.1392817040 **(27)**

> $plot([xcrash(t, .2), xcrash(t, .3), xcrash(t, .4)], t = -\mathrm{Pi}..\mathrm{Pi}, dist = 0..12)$



So all is good, right? What if R is really small?

> $xcrash(0, .05)$

19.9282874245 **(28)**

> $xcrash(0, 0)$

20.0000000000 **(29)**

Do you believe these? You shouldn't.

> $solfunc(0, .1)(20)$

$[t = 20.0000000000, clock(t) = 20.0000000000, \theta(t) = -0.1046720285, v(t) = 0.9971563773, x(t) = 19.7921201575, y(t) = 0.0307219138]$ **(30)**

> $solfunc(0, .05)(20)$

$[t = 20.0000000000, clock(t) = 20.0000000000, \theta(t) = -0.0611394053, v(t) = 0.9991947195, x(t) = 19.9282874245, y(t) = 1.0066520645]$ **(31)**

The problem is that you need t>20 to hit the ground for R=.1 or less.
We can fix this by increasing the time to look for the crash, but... It might make more sense to try some time, if it didn't work, go a little more, etc. But let's leave that alone for now....

Next time, we start doing cryptography stuff.