> *Alexander Sage*

> *Project* 2

> 04 − 24 − 2012

>

This maple worksheet is to examine the flight of a glider.
The two main differential equations in this study are

> $D(v)(t) = -\sin(\text{theta}(t)) - 0.1 \cdot v(t)^2$

$$D(v)(t) = -\sin(\theta(t)) - 0.1\, v(t)^2 \quad \textbf{(1)}$$

so the change in velocity is dependant on the angle of the nose of the glider (causes lift), and the friction (drag coefficient set equal to 0.1)

> $D(\text{theta}(t)) = v(t) - \dfrac{\cos(\text{theta}(t))}{v(t)}$

$$D(\theta(t)) = v(t) - \frac{\cos(\theta(t))}{v(t)} \quad \textbf{(2)}$$

the change in theta is dependant on there being enough velocity with respect to the horizontal to apply a great enough fricitonal force that the lift continues to increase than angle of the nose.
The change in the x (horizontal) and y (vertical) direction of the rocket can be defined relative to v and theta as

> $D(y(t)) = v(t) \cdot \sin(\text{theta}(t))$

$$D(y(t)) = v(t)\sin(\theta(t)) \quad \textbf{(3)}$$

> $D(x(t)) = v(t) \cdot \cos(\text{theta}(t))$

$$D(x(t)) = v(t)\cos(\theta(t)) \quad \textbf{(4)}$$

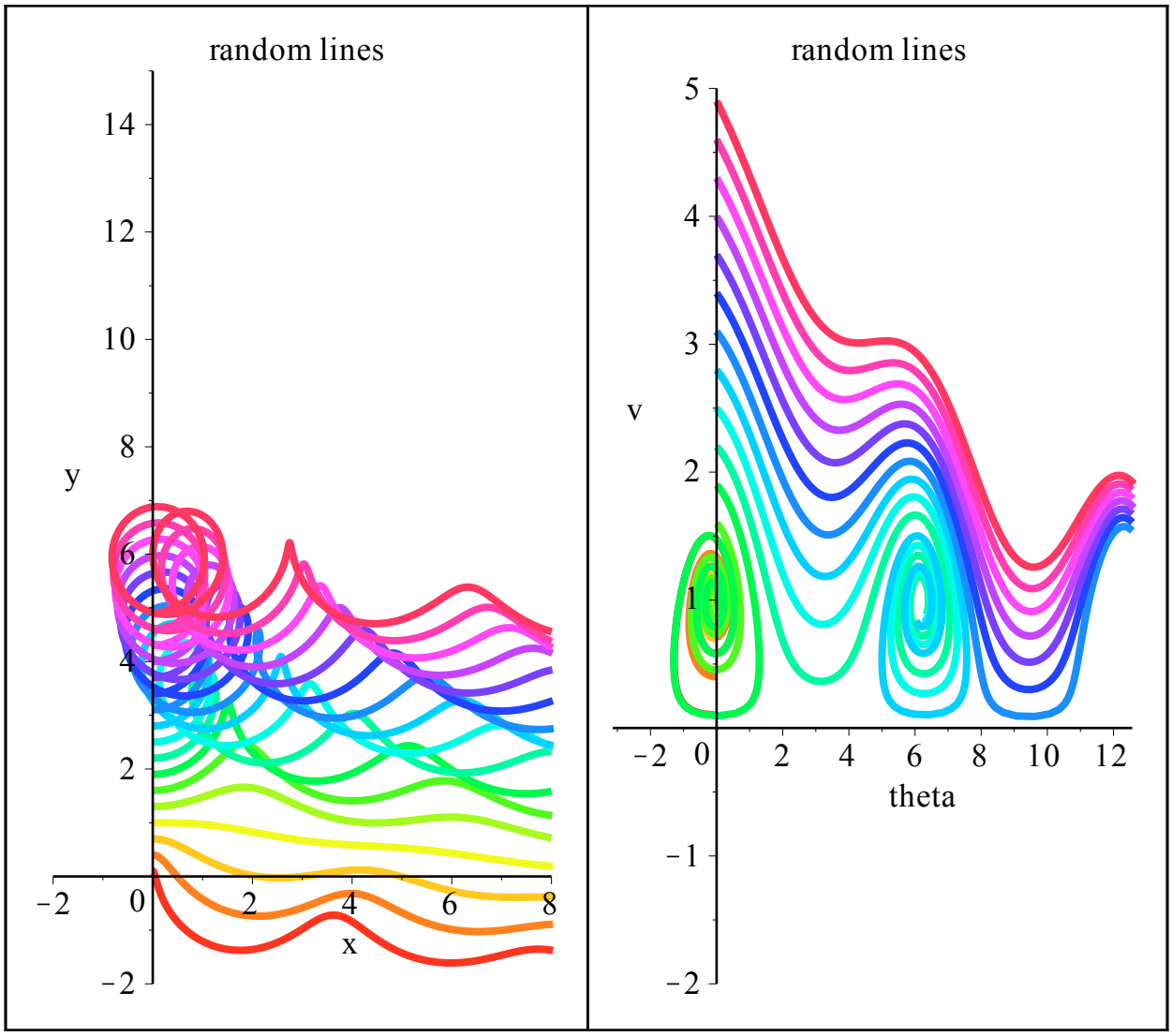
The system of differential equations are put together and will be evaluated to find initial conditions of the glider that cause it to make approximately one/two loops, travel 20 units of distance, fly the furthest (x) distance, and land gently to the ground.

> $DE := [D(v)(t) = -\sin(\text{theta}(t)) - 0.1 \cdot v(t)\text{^}2, D(\text{theta})(t) = v(t) - \cos(\text{theta}(t))/v(t), D(x)(t) = v(t) \cdot \cos(\text{theta}(t)), D(y)(t) = v(t) \cdot \sin(\text{theta}(t))]$:

The first step will be to find the initial velocities of the glider (with initial angle fixed at 0) at which it makes its first and second loop respectively. (The initial x(t) and y(t) are irrelevant for this part of the analysis because it doesn't matter where the glider is in space as long as it makes a single loop. The values of x(t) and y(t) can however be used to show that the glider has made one/two succesful loop(s))

A random plot of varying values must be made to zoom in on the correct ones.

```
> guess := [theta(t), v(t), x(t), y(t)], t = 0..15, theta = -Pi..4·Pi, v = -2..5, x = -2..8, y = -2
      ..15,
      [seq([theta(0) = 0, x(0) = 0, y(0) = i, v(0) = i], i = 0.1..5, 0.3)], linecolor
      = [seq(COLOR(HUE, i), i = 0..1, .06)], stepsize = 0.05 :
  display(array([DEplot(DE, guess, scene = [x, y]), DEplot(DE, guess, scene = [theta, v])]),
      title = "random lines")
```



The range of velocities that result in a perticular swirl is the range of velocities that correspont to that corresponding number of loops

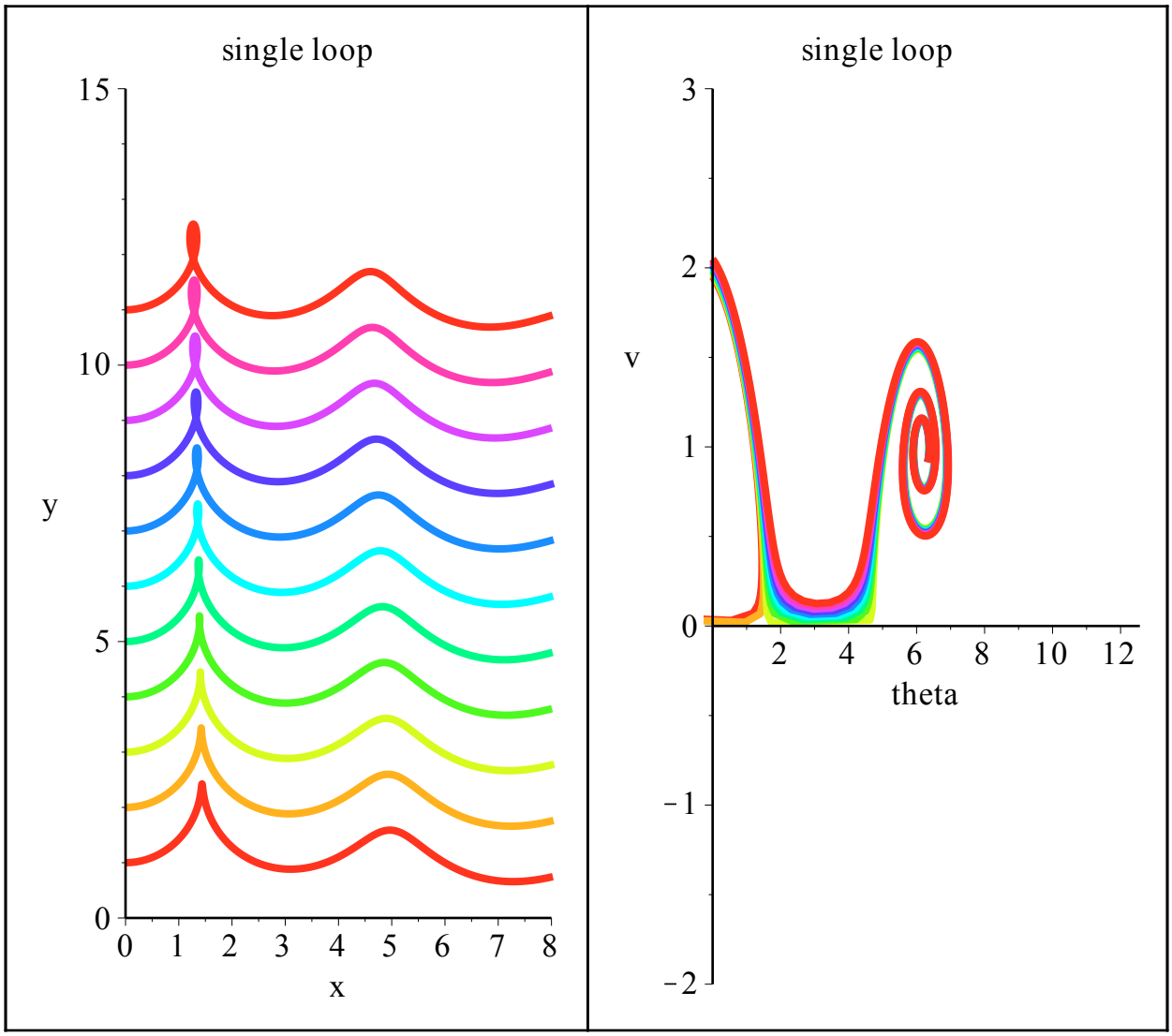
'rang' below describes the range of initial conditions that the two plots show
the different lines represent initial values of velocity starting at 1.95 and increasing to 2.05 with every 0.01.

```
> rang := [theta(t), v(t), x(t), y(t)], t = 0..15, theta = 0..4·Pi, v = -2..3, x = 0..8, y = 0..15,
      [seq([theta(0) = 0, x(0) = 0, y(0) = (i - 1.95)·100 + 1, v(0) = i], i = 1.95..2.05, 0.01)],
       linecolor = [seq(COLOR(HUE, i), i = 0..1, .1)], stepsize = 0.05 :
```

It can be seen on the graph on the left that the glider makes approximately one loop
On the graph on the right the theta value can be seen to have a maximum of about 7 (slightly over 2·Pi = 6.283185308; the minimum value that corresponds to one full circle)

```
> with(DEtools) :
  with(plots) :
```

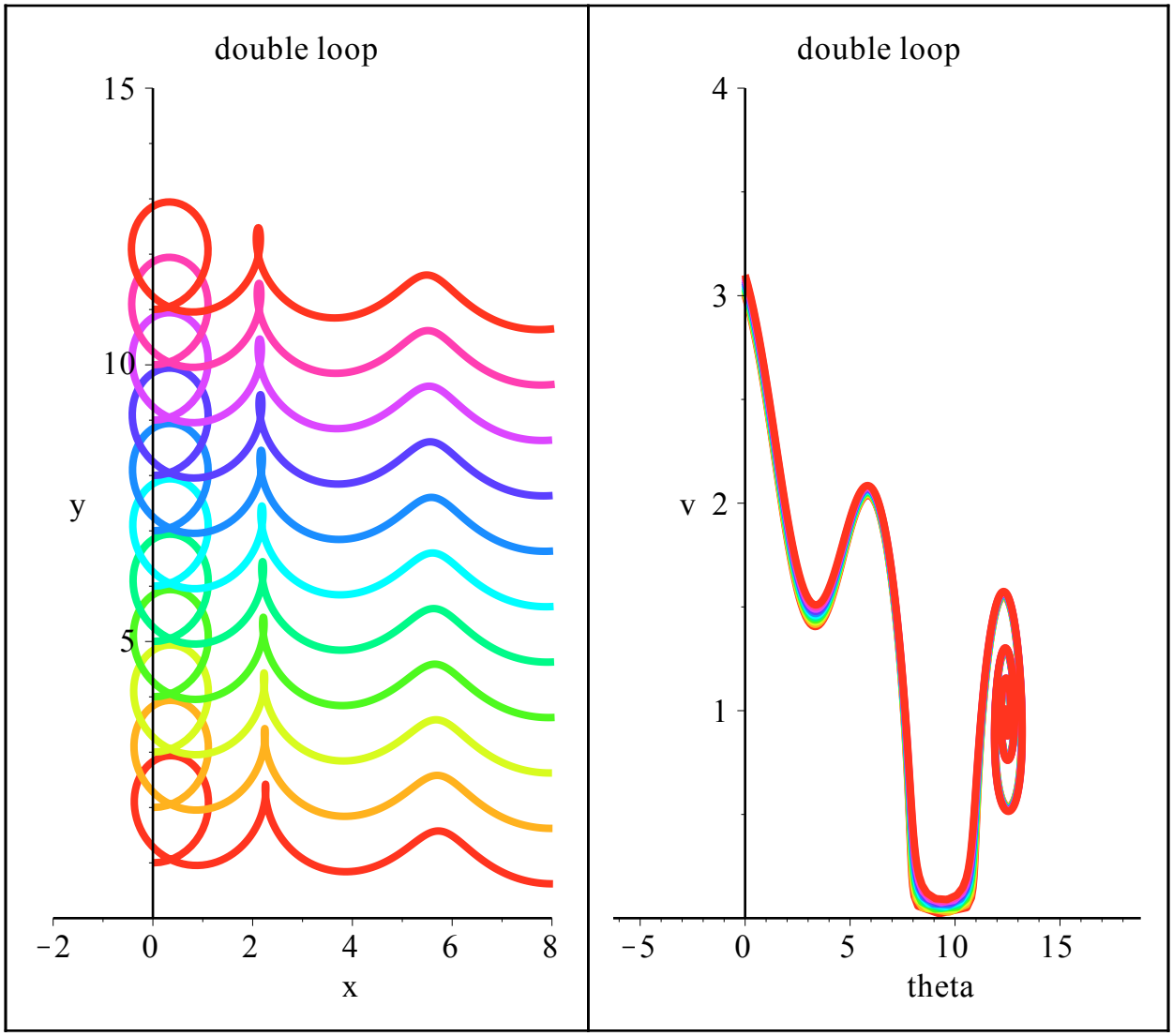
```
> display(array([DEplot(DE, rang, scene = [x, y])], DEplot(DE, rang, scene = [theta, v])]),
      title = "single loop")
```



With an error of 0.05 the initial velocity for a single loop can be estimated to be v(0)=2 (the middle line)

'rang2' represents the range of initial conditions under which the second loop emerges
the lines start with an initial velocity of 3.0 and increase with 0.01 to 3.1

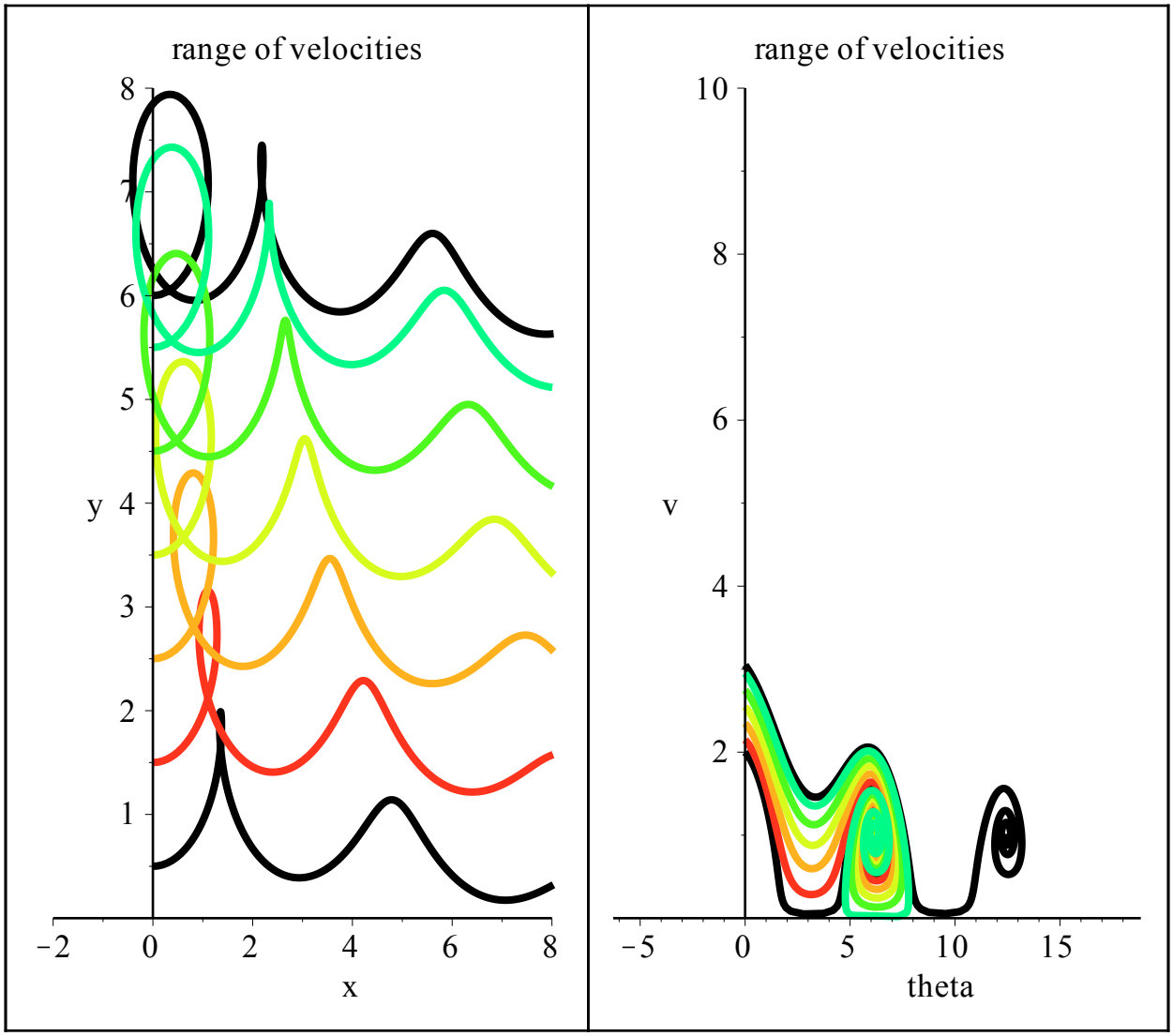
```
> rang2 := [theta(t), v(t), x(t), y(t)], t = 0..20, theta = -2·Pi..6·Pi, v = 0..4, x = -2..8, y = 0
      ..15,
      [seq([theta(0) = 0, x(0) = 0, y(0) = (i - 3)·100 + 1, v(0) = i], i = 3.0..3.1, 0.01)],
       linecolor = [seq(COLOR(HUE, i), i = 0..1, .1)], stepsize = 0.05 :
> display(array([DEplot(DE, rang2, scene = [x, y]), DEplot(DE, rang2, scene = [theta, v])])),
      title = "double loop")
```



The theta can be seen to vary around 4·Pi=12.56637062
The initial velocity can be estimated in the middle at v(0)=3.05 for two loops

For the graph below:
In black the estimated initial conditions for one (v(0)=2.0) and two (v(0)=3.05) loops are plotted
the lines in between are arbitrary increasing initial values of velocity to show the transition from one loop to two as the initial velocity increases.

```
> rang3 := [theta(t), v(t), x(t), y(t)], t = 0..20, theta = -2·Pi..6·Pi, v = 0..10, x = -2..8, y = 0
      ..8,
      [[theta(0) = 0, x(0) = 0, y(0) = 0.5, v(0) = 2], [theta(0) = 0, x(0) = 0, y(0) = 6, v(0)
      = 3.05], seq([theta(0) = 0, x(0) = 0, y(0) = (i - 2.05)·5 + 1, v(0) = i], i = 2.15..3.05,
      0.2)], linecolor = [black, black, seq(COLOR(HUE, i), i = 0..1, .1)], stepsize = 0.05 :
> display(array([DEplot(DE, rang3, scene = [x, y]), DEplot(DE, rang3, scene = [theta, v])]),
      title = "range of velocities")
```



For all the values in between v(0)=2.0 and v(0)=3.05 the the glider will make exactly one loop

For a given distance, say 20, initial conditions and initial velocities can be found that result in such an x value (with some error).

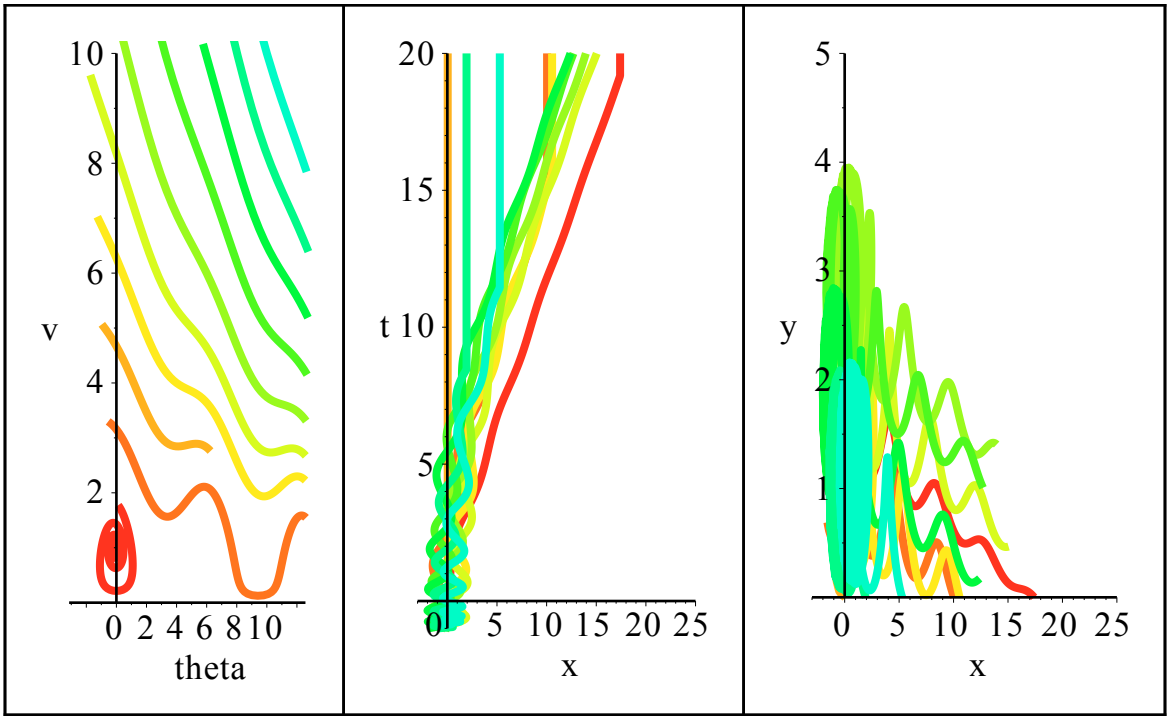
To examine the system as it goes to a specific distance, the system needs to end once the glider hits the ground

The previous system of differential equations is changed so that once the y value equals 0 (hits the ground), the time derivative of all variables is set to 0.

```
> crash := [ D(theta)(t) = piecewise( y(t) > 0, v(t) - cos(theta(t))/v(t), 0),
        D(v)(t)     = piecewise( y(t) > 0, -sin(theta(t)) -0.1· v(t)^2, 0),
        D(x)(t)     = piecewise( y(t) > 0, v(t) * cos(theta(t)), 0),
        D(y)(t)     = piecewise( y(t) > 0, v(t) * sin(theta(t)), 0)] :
```

The graph below is simply random initial velocities and initial angles each increasing at the same rate from 1 to 10 with a step size of 1

```
> condish := [theta(t), v(t), x(t), y(t)], t = -1..20,
    theta = -Pi..4 * Pi, v = 0..10, x = -3..25, y = 0..5,
    [seq([theta(0) = i, v(0) = i, x(0) = 0, y(0) = 2], i = 1..10, 1)],
    linecolor = [seq(COLOR(HUE, i), i = 0..1, .05)], stepsize = 0.05 :
  display( array( [ DEplot(crash, condish, scene = [theta, v]),
            DEplot(crash, condish, scene = [x, t]), DEplot(crash, condish, scene = [x, y]) ]))
```



To get actual values, a function is defined as 'target' with variables for initial theta and velocity ('thet' and 've')
the function is the second system of differential equations 'crash' solved with initial x(0)=0 and y(0)=2 and yet to be determined initial values for theta and velocity

```
> target := dsolve( {op(crash), theta(0) = thet, v(0) = ve, x(0) = 0, y(0) = 2}, numeric,
      parameters = [thet, ve])
```

$$target := \mathbf{proc}(x\_rkf45) \ \ldots \ \mathbf{end\ proc} \tag{5}$$

when solved it looks like this

```
> target(parameters = [thet = 0, ve = 9])
```

$$[thet = 0., ve = 9.] \tag{6}$$

```
> target(3)
```

$$[t = 3., \theta(t) = 13.0330654437293, v(t) = 2.69096992226788, x(t) = 0.868051321702848, y(t) = 2.09288904484495] \tag{7}$$

A code has been written to loop over all possible initial values of velocity and angle, and print to the screen the values where the final x value comes within a certain error of 20

```
> bullseye := proc( f, t, vl, vh, thel, theh, eps, i )
      #the program takes in a function (f), an initial time (t), a starting

      #velocity (vl), and ending velocity vh, a starting angle ( thel), an

      #ending angle (theh), an error for the x value to be accepted (eps)
                                                        #a loop index (i)
   local vel, ang, time, num;
   vel := vl;
  ang := thel;
   time := t;
   num := 1;
  print("number,velocity, angle, time, x, y");     #writes a headline to interpret results
  while(vel ≤ vh) do
      #starts the loop and runs the following code until the velocity
                                                    #parameter is equal the ending velocity
   f( parameters = [ve = vel]);                     #this sets the velocity parameter
       while(ang ≤ theh)do
      #starts the code to loop over 'thet' until the ending angle
          f( parameters = [thet = ang]);            #this sets the theta parameter
            while(rhs( f(time)[5]) > 0) do
      #this loop increases the time by steps of i until the
              time := time + i;                     #glider crashes and the y value is no
           end do;
            if (abs(rhs( f(time)[4]) − 20) < eps) then
      #if the glider has traveled a distance of 20

      #within an error equal to eps the program prints out the
                print(num, vel, ang, time, rhs( f(time)[4]), rhs( f(time)[5]));
      #parameters used
                num := num + 1;
            end if;
          time := 0;              #the time is reset to 0 to be evaluated at a different angle
          ang := ang + i;
      #the angle parameter is increased by i and the new distance is found
       end do;
    ang := thel;
      #the angle is reset to the initial angle to be looped over, and the velocity
    vel := vel + i;
      #is increased by i . the whole code is then repeated for a new velocity parameter
  end do;
  return("fin!");
  end:
```

the code 'bullseye' takes in the function target and other values to use to evaluate the answer

```
> bullseye(target, t, vl, vh, thetl, theth, eps, i)
```

"number,velocity, angle, time, x, y" **(8)**

The initial angles can only range from 0 to 2*Pi (since 0=2Pi=4Pi...), and the initial velocity must have a max initial velocity (randomly chosen to be equal to 10)

```
> evalf(2·Pi)
```

6.283185308 **(9)**

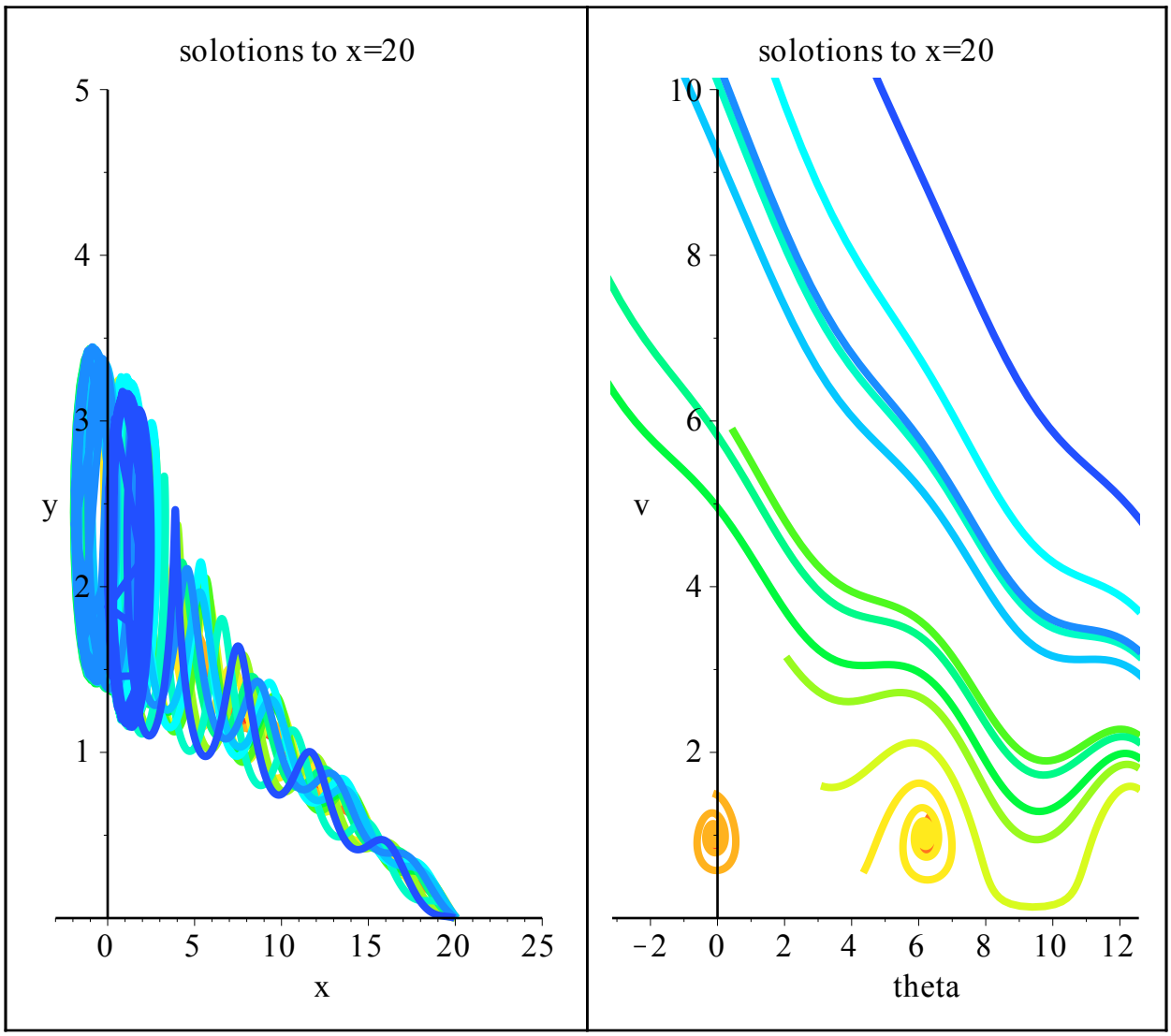
```
> bullseye(target, 0, 0.1, 10, 0, 6.5, 0.05, 0.1):
```

using the code - 14 different sets of parameters that were found to crash at a distance of 20 within an error of 0.05

```
> par1 := [v(0) = 1, theta(0) = 0.1, x(0) = 0, y(0) = 2]:
  par2 := [v(0) = 1, theta(0) = 6.5, x(0) = 0, y(0) = 2]:
  par3 := [v(0) = 1.1, theta(0) = 0.5, x(0) = 0, y(0) = 2]:
  par4 := [v(0) = 1.4, theta(0) = 5.3, x(0) = 0, y(0) = 2]:
  par5 := [v(0) = 2.0, theta(0) = 5.1, x(0) = 0, y(0) = 2]:
  par6 := [v(0) = 2.7, theta(0) = 5.0, x(0) = 0, y(0) = 2]:
  par7 := [v(0) = 3.8, theta(0) = 5.0, x(0) = 0, y(0) = 2]:
  par8 := [v(0) = 4.3, theta(0) = 1.1, x(0) = 0, y(0) = 2]:
  par9 := [v(0) = 5.1, theta(0) = 1.1, x(0) = 0, y(0) = 2]:
 par10 := [v(0) = 6.2, theta(0) = 4.9, x(0) = 0, y(0) = 2]:
 par11 := [v(0) = 7.4, theta(0) = 5, x(0) = 0, y(0) = 2]:
 par12 := [v(0) = 8.2, theta(0) = 1.1, x(0) = 0, y(0) = 2]:
 par13 := [v(0) = 9.2, theta(0) = 1.1, x(0) = 0, y(0) = 2]:
 par14 := [v(0) = 10, theta(0) = 4.9, x(0) = 0, y(0) = 2]:
```

all these parameters found are plotted below, as can be seen they all land very close to 20 and start with many different initial velocities and angles.

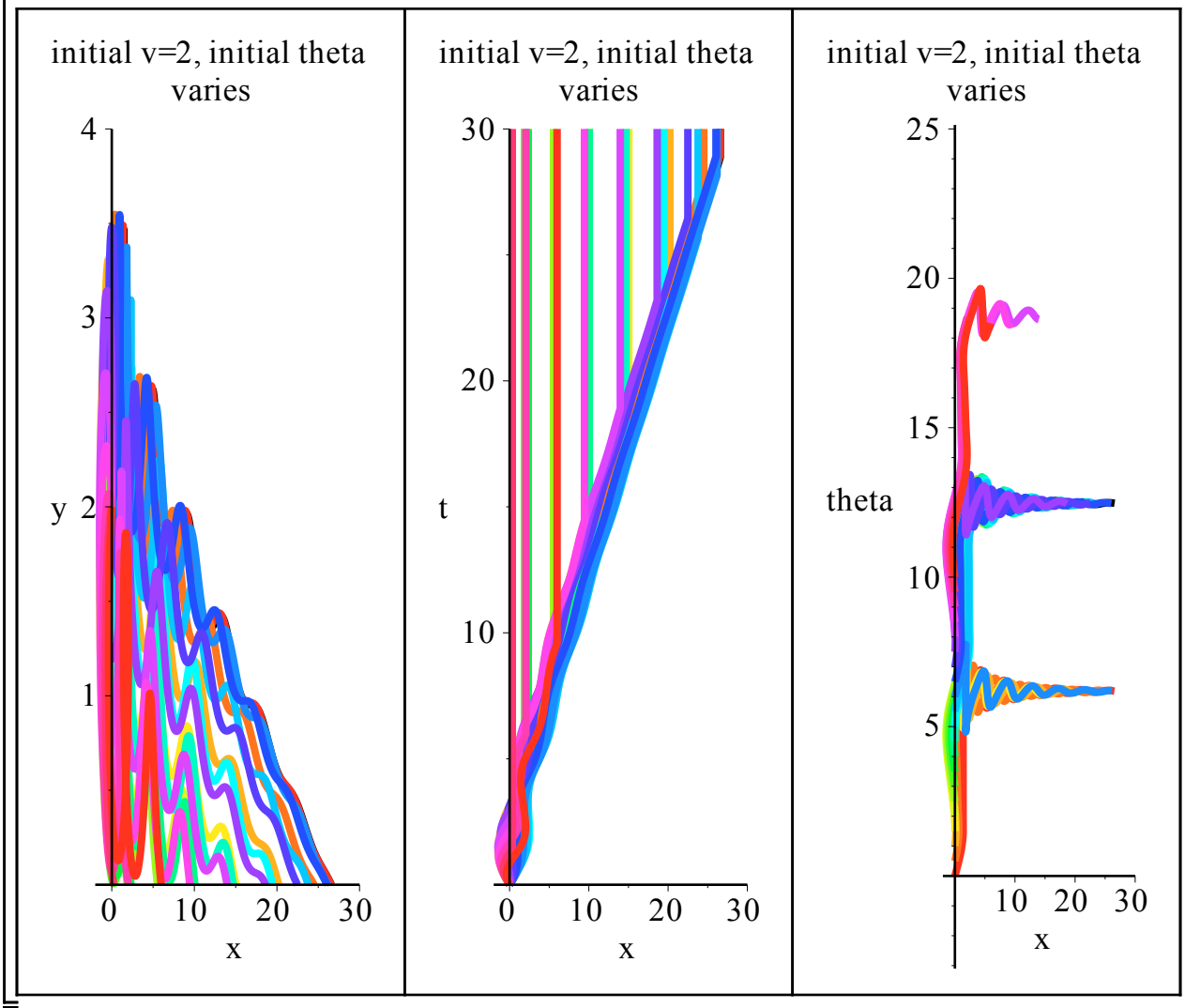
```
> pars := [theta(t), v(t), x(t), y(t)], t = -1..30,
    theta = -Pi..4*Pi, v = 0..10, x = -3..25, y = 0..5,
    [par1, par2, par3, par4, par5, par6, par7, par8, par9, par10, par11, par12, par13, par14],
    linecolor = [seq(COLOR(HUE, i), i = 0..1, .05)], stepsize = 0.05 :
  display(array([DEplot(crash, pars, scene = [x, y], title = "solotions to x=20"),
      DEplot(crash, pars, scene = [theta, v], title = "solotions to x=20")]))
```



With fixing v(0)=2, x(0)=0 and y(0)=2 theta is varied and for each individual value of theta the glider goes a different distance before crashing. By ranging over all possible initial values of theta, the parameters for the glider to fly a maximum distance can be found.

The plot below gives some insight into what values should be investigated. the graph on the left shows that there must be a glider that goes over 20. the graph in the middle shows that the gliders that travel the farthest x all go over a value of t=25. the graph on the right shows the initial values of theta only range from 0 to 2*Pi

```
> conditions := crash, [theta(t), v(t), x(t), y(t)], t = 0..30,
    theta = -Pi..8·Pi, v = 0..10, x = -2..30, y = 0..4,
    [[theta(0) = 6.22, v(0) = 2, x(0) = 0, y(0) = 2], seq([theta(0) = i, v(0) = 2, x(0) = 0, y(0)
      = 2], i = 0..10, .5)],
    linecolor = [black, seq(COLOR(HUE, i), i = 0..1, .05)], stepsize = 0.05 :
> display(array([DEplot(conditions, scene = [x, y]), DEplot(conditions, scene = [x, t]),
    DEplot(conditions, scene = [x, theta])]), title = "initial v=2, initial theta varies")
```



>

A code called 'nocrashy' is written to evaluate parameters and find the set that results in the glider flying the farthest

```
> nocrashy := proc( f, init, a, b, epsilon)
      #takes in a function (f), initial value of time (init), a range of values to
                                                  #loop theta over ( a to b), and a loop index (epsilon)
   local   ans, gxt, tempx, d, time, ang, ltime;
   time := init;                                  #time is set to start at init
   ang := a;                                     #the angle is set to start at a
   d := 0;                                      #d is the greatest x value obtained for any value of theta
   ltime := 0;
      #ltime is the amount of time the glider is in the air to travel 'd'
   gxt := 0;                                    #gxt is the greatest x value for a specific theta value
   while ( a ≤ ang ≤ b)do                       #the loop of theta from a to b starts
   f ( parameters = [thet = ang ]);             #the parameter for theta is set to 'ang'
     while (rhs( f (time ) [5]) > 0) do
      #as long as the glider is above the ground this loop continues
       tempx := rhs( f (time ) [4]);
      #tempx is the x distance assigned for the function at time 'time'
          if (tempx > gxt ) then
      #if 'tempx' is the greates x distance found for that theta, 'gxt' is redifined
            gxt := tempx;                       #to equal 'tempx'
          fi;
       time := time + epsilon;
      #time is increased, the loop is restarted and the new x distance is evaluated
     end do;
     if (gxt > d ) then
      #if the greatest x value for that theta is the greatest x value for any other theta
     d := gxt;                                  #evaluated, then 'd' is set equal to 'gxt'
     ans := ang;                                #the associated initail angle and final time are saved with 'd'
     ltime := time;
     fi;
   ang := ang + epsilon;
      #the initial angle is increased by epsilon and the whole loop restarts.
   end do;
   return ("initial angle=", ans, "distance traveled=", d, "time taken", ltime ) ;
      #the answer is printed to the screen
   end:
```

inv2 is function with proper fixed initial conditions, that can vary the initial theta value

```
> inv2 := dsolve( {op(crash), theta(0) = thet, v(0) = 2, x(0) = 0, y(0) = 2}, numeric,
      parameters = [thet])
                                  inv2 := proc(x_rkf45) ... end proc                    (10)
```

an initial value of t=25, a range of theta= 0 to 7 and a low precision is chosen based on the graph

```
> nocrashy(inv2, 25, 0, 7, 1)
        "initial angle=", 0, "distance traveled=", 25.6144591714577, "time taken", 29   (11)
```

from this run it is apparent that the initial theta value has to be close to 0
in terms of radians 0 = 2·Pi , zoom in around 2Pi

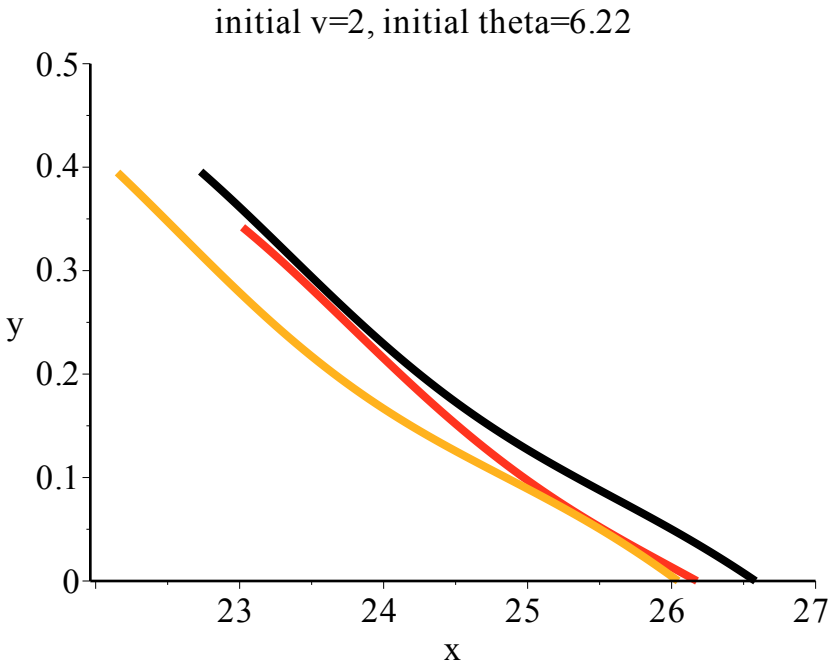
so we run it again but make the range smaller (between 5 and 7) and we increase the precision to one hundredth of a radian

```
> nocrashy(inv2, 25, 5, 7, 0.01)
```

"initial angle=", 6.22, "distance traveled=", 26.5814542337952, "time taken", 28.88 **(12)**

The initial angle =6.22 is plotted below in black along with two other lines that corresponds to initial angles less than and greater than 6.22

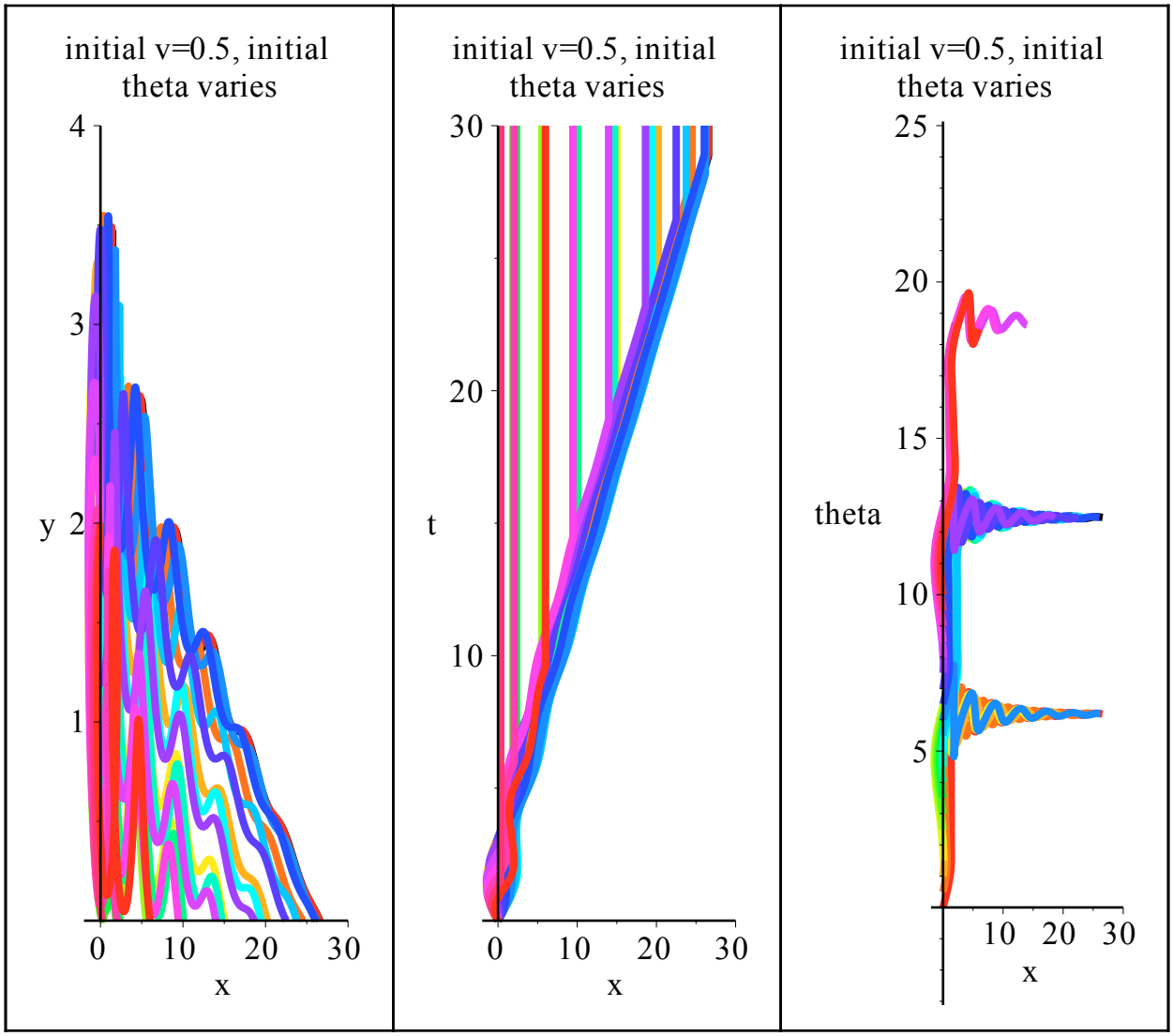
```
> DEplot( crash, [theta(t), v(t), x(t), y(t) ], t = 25 ..30,
    theta = -Pi ..4 * Pi, v = 0 ..10, x = 22 ..27, y = 0 ..0.5,
    [ [theta(0) = 6.22, v(0) = 2, x(0) = 0, y(0) = 2 ], seq( [theta(0) = i, v(0) = 2, x(0) = 0, y(0)
      = 2 ], i = 6 ..6.5, .5) ],
    linecolor = [black, seq(COLOR(HUE, i), i = 0 ..1, .1) ], stepsize = 0.05,
    scene = [x, y ], title = "  initial v=2, initial theta=6.22")
```



>

Now the initial velocity will be changed from 2 to 0.5 and the same process will be taken to find the max x value

```
> conditions2 := crash, [theta(t), v(t), x(t), y(t)], t = 0..30,
    theta = -Pi..8*Pi, v = 0..10, x = -2..30, y = 0..4,
    [seq([theta(0) = i, v(0) = 0.5, x(0) = 0, y(0) = 2], i = 0..10, 1)], linecolor
      = [seq(COLOR(HUE, i), i = 0..1, .05)], stepsize = 0.05 :
> display(array([DEplot(conditions, scene = [x, y]), DEplot(conditions, scene = [x, t]),
    DEplot(conditions, scene = [x, theta])]), title = "initial v=0.5, initial theta varies")
```




```
>
> inv5 := dsolve({op(crash), theta(0) = thet, v(0) = 0.5, x(0) = 0, y(0) = 2}, numeric,
    parameters = [thet])
```

$$inv5 := \mathbf{proc}(x\_rkf45) \ \ldots \ \mathbf{end\ proc} \qquad \textbf{(13)}$$

```
> nocrashy(inv5, 0, 0, 7, 1)
```

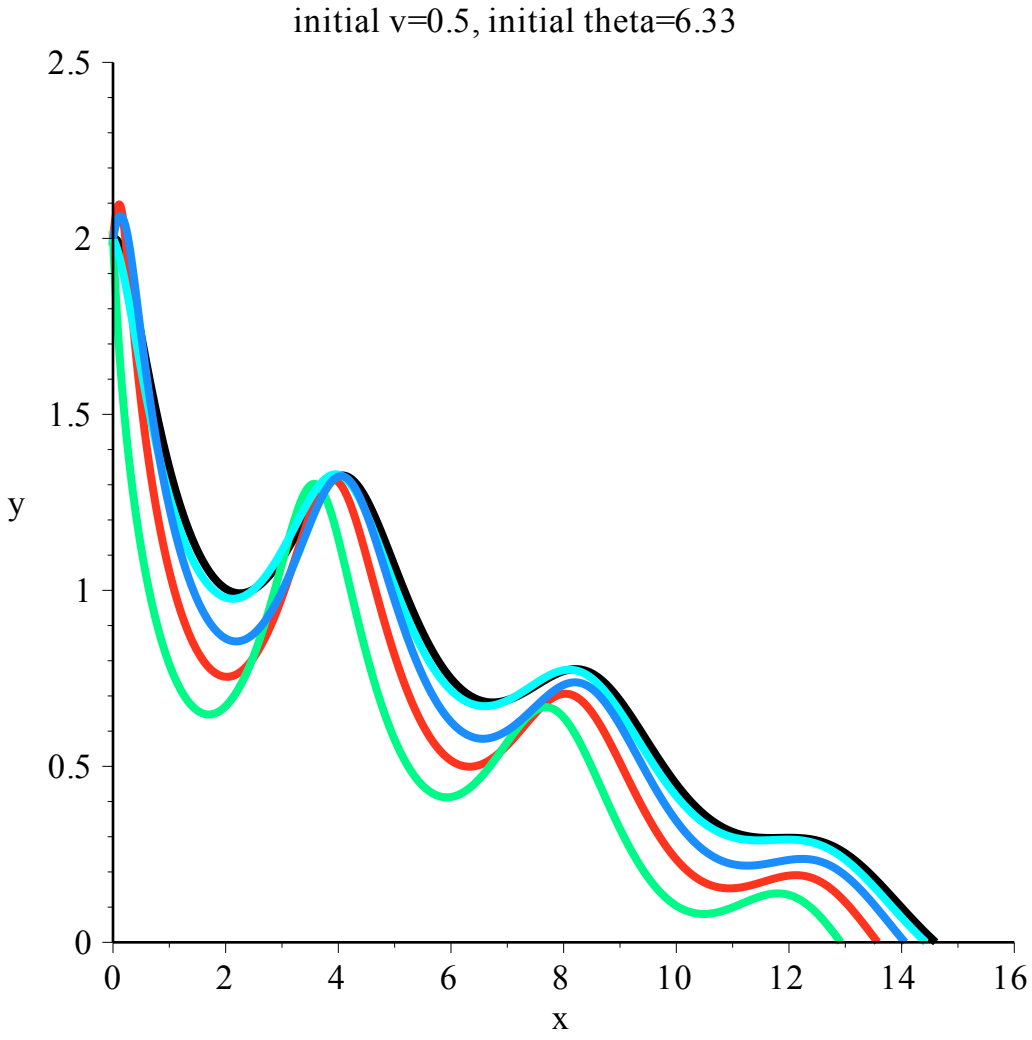
"initial angle=", 0, "distance traveled=", 14.3950832532543, "time taken", 16 **(14)**

even when the initial velocity is changed from 2 to 0.5, the initial angle is 0
so we increase the precision of the code, run it again, narrow the range and put the more precise initial angle into the plot with other nearby initial values.

```
> nocrashy(inv5, 0, 5, 7, 0.01)
```

"initial angle=", 6.32, "distance traveled=", 14.6047011482156, "time taken", 15.23 **(15)**

```
> DEplot( crash, [theta(t), v(t), x(t), y(t)], t = 0..30,
  theta = -Pi..4*Pi, v = 0..10, x = 0..16, y = 0..2.5,
  [[theta(0) = 6.32, v(0) = 0.5, x(0) = 0, y(0) = 2], seq([theta(0) = i, v(0) = 0.5, x(0) = 0,
    y(0) = 2], i = 1..7, 1)],
  linecolor = [black, seq(COLOR(HUE, i), i = 0..1, .1)], stepsize = 0.05,
  scene = [x, y], title = "initial v=0.5, initial theta=6.33")
```



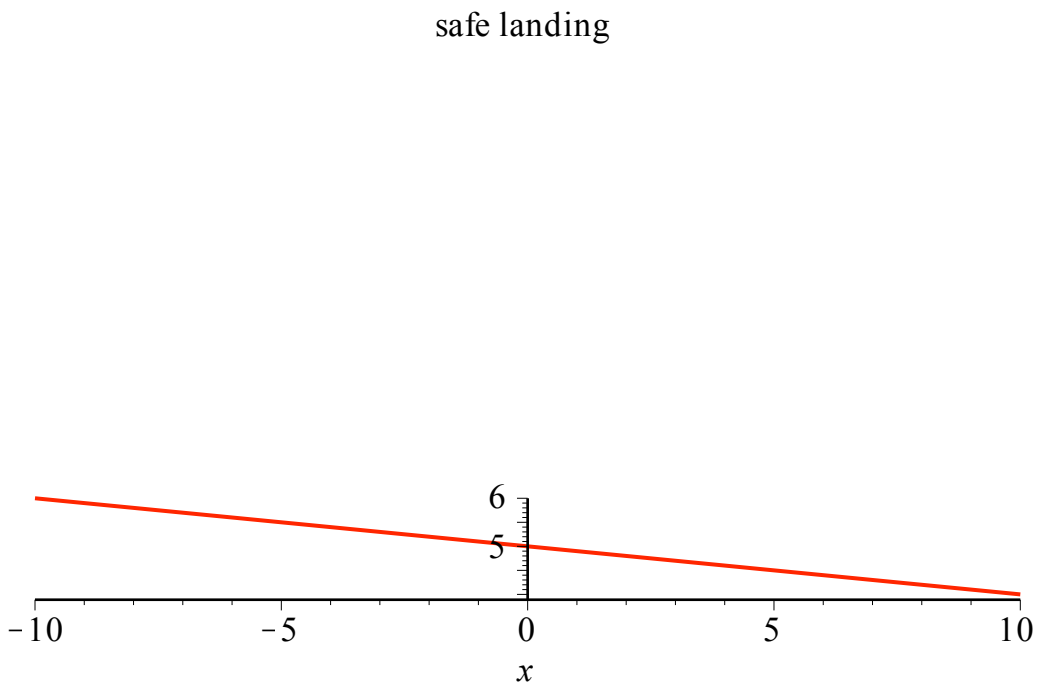
the initial parameters found from 'nocrashy' (in black) can be seen to travel the farthest

The next analysis of the glider is to decide if, with equations 'crashy', the glider can land smoothly on the ground without necessarily crashing.

>

The graph here has a line with an arbitrary y-intercept and with a slope of -0.1. The purpose is to take a guess as to what the path of glider would look like if it were to fly straight into a landing strip. The glider, when close to y=0, should have a slope similar or smaller than this one shown, or the nose of the glider may hit the ground too hard.

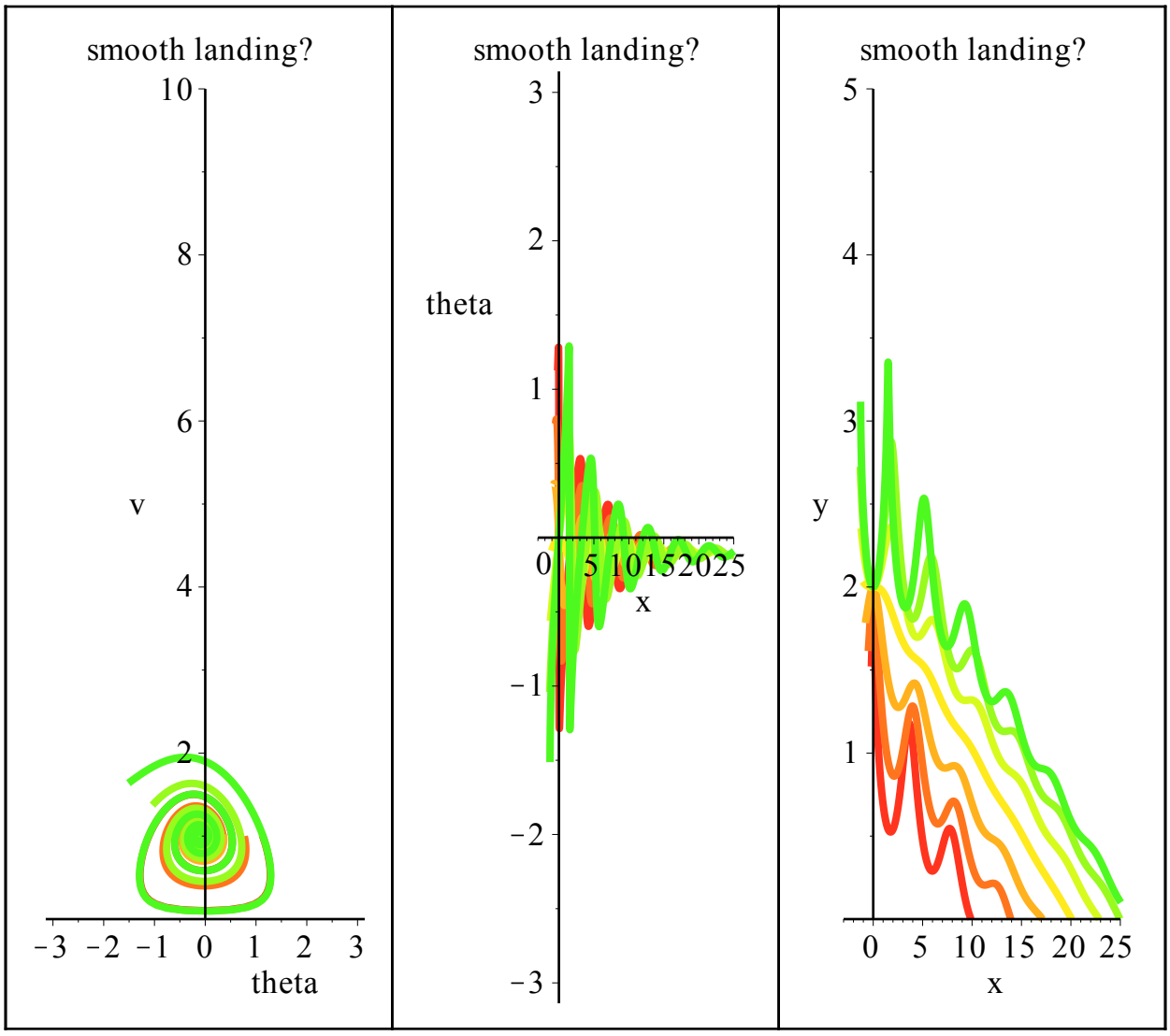
```
> plot( -0.1 ·x + 5, scaling = constrained, title = "safe landing" )
```



>

A set of initial conditions are run where velocity ranges from 0 to 2.1. After the glider passes the boundary between no loops and one, the general behavior of the graph repeats and isn't worth analyzing.

```
> condish := [theta(t), v(t), x(t), y(t)], t=-1..30,
    theta=-Pi.. Pi, v=0..10, x=-3..25, y=0..5,
   [seq([theta(0)=0, v(0)=i, x(0)=0, y(0)=2], i=0.1..2.1, 0.3)],
   linecolor=[seq(COLOR(HUE, i), i=0..1, .05)], stepsize=0.05 :
  display(array([ DEplot(crash, condish, scene=[theta, v]),
                  DEplot(crash, condish, scene=[x, theta]), DEplot(crash, condish, scene=[x, y]) ]
      ), title="smooth landing?")
```



A code called 'smooth' is written to find initial conditions that result in the glider hitting the ground when it is almost horizontal

```
> smooth := proc( f, t, vl, vh, thel, theh, eps, i, slow)
  local  vel, ang, time, num, slope, speed;
  vel := vl;
      #the code was written almost exactly as bullseye was but
  ang := thel;
      #a value called 'slow' is passed in that represents the maximum velocity
  time := t;                                  #the glider can have when landing
  num := 1;
 print("number,initial velocity, initial angle, time, slope, velocity");
 while(vel ≤ vh) do
  f( parameters = [ve = vel]);
       while(ang ≤ theh)do
      #the code evaluates all possible values of theta and velocity
         f( parameters = [thet = ang]);
            while(rhs( f(time)[5]) > 0) do
      #finds the time for each when the glider hits the ground
              time := time + i;
            end do;
             slope := tan(rhs( f(time)[2]));
      #takes the slope of the glider at the instant of contact
             speed := rhs(f(time)[3]);            #takes the velocity at the time of impact
             if (abs(slope) ≤ eps and slope ≤ 0 and 0 ≤ speed ≤ slow) then
      #if both slope and velocity are good
                 print(num, vel, ang, time, slope, speed);
      #enough values for landing, then the conditions are
                 num := num + 1;                                  #printed to the screen
             end if;
          time := 0;
          ang := ang + i;
       end do;
    ang := 0;
    vel := vel + i;
  end do;
  return("fin!");
  end:
>
```

the same function used earlier is used again

```
> target := dsolve( {op(crash), theta(0) = thet, v(0) = ve, x(0) = 0, y(0) = 2}, numeric,
      parameters = [thet, ve])
```

$$target := \mathbf{proc}(x\_rkf45) \ \ldots \ \mathbf{end\ proc} \qquad (16)$$

the code takes in values as shown below

```
> #smooth(function, initial time, vl, vh, thel, theh, eps, i, slow)
```

vl, vh is the velocity range
thel, theh is the theta range
eps is the maximum slope of the glider for the found values. the closer eps is to 0 the more horizontal the glider is on impact
i is the loop index for the time, angle and velocity
slow is the highest velocity the glider may have when hitting the ground.

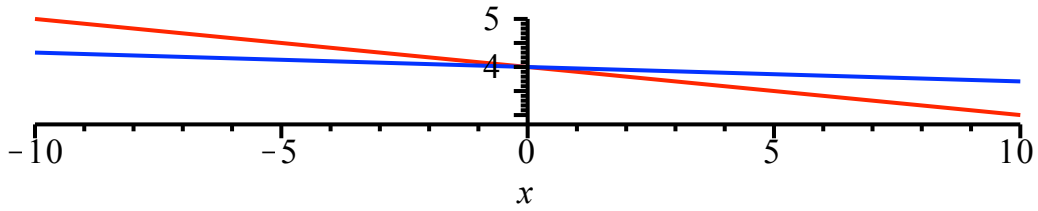
```
>
> smooth(target, 0, 2, 3.1, 0, 6.5, 0.1, 0.2, 1)
```

the results found for either range are similar enough to support the assumption that no further values of velocity need be analized.

```
> smooth(target, 0, 0.1, 2.1, 0, 6.5, 0.1, 0.2, 1)
```

When the program is run for both ranges, over 45 values are found that satisfy the conditions that the slope is less than -0.1 and that the maximum velocity is less than 1. The glider doesn't seem to land with many values of velocity lower than 0.9, but that seems sufficiently low. The slope upon landing tended to be around 0.07, but did range from roughly 0.1 (shown in the graph below in red) and 0.03 (shown in blue)

```
> display({plot(-0.1 x + 4), plot(-0.03·x + 4, color = blue)}, scaling = constrained)
```



Assuming that the glider has tires (some protective buffer) that can create friction with the ground, and has a long enough landing strip to slow down, then the many values found - correspond to a successful landing. The glider can land without *crashing*.