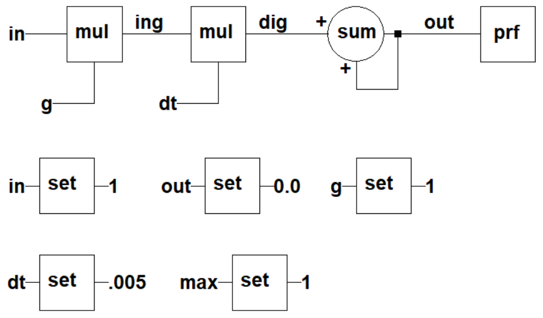
# Experiments with Integrators

Let’s start by building a simple first order (Euler) integrator. The function is built using simple math blocks. First the input is multiplied by gain g. Next, the area under the input times gain is calculated by multiplying by dt (the step time). Length times width is the rectangular area under the input curve.

Integrators sum or accumulate the area after each dt time step. The summing or addition block is used for this purpose. The new area calculation (dig = dt*g*in) is used as one input and the other is the last output. Thus, out(t) = out(t-dt) + dig.

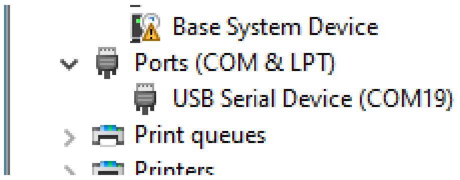
The function performed is $g * \int_0^t in\ \delta t + c$:

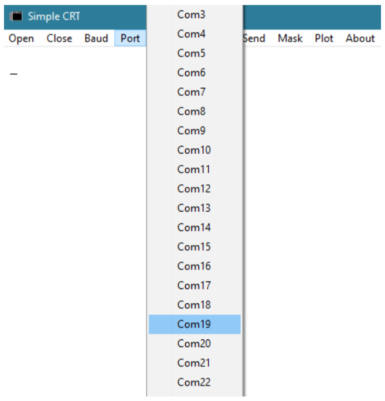
The initial condition c is the output set to zero at time t=0. It could be initialized to any value.

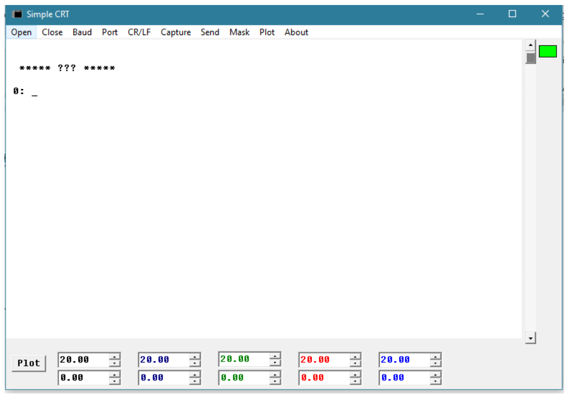


```
clr
mul in g ing
mul ing dt dig
sum dig out out
prf out
end
set in 1
set out 0.0
set g 1
set dt .005
set max 1
```

Use simpleCRT.exe to open a serial connection to your board. If you don’t know your serial port number, go to Settings, search Device Manager then run Device Manager. Click on Ports (COM & LPT). You will find your serial port COM number.







```
***** ??? *****
0: _
```

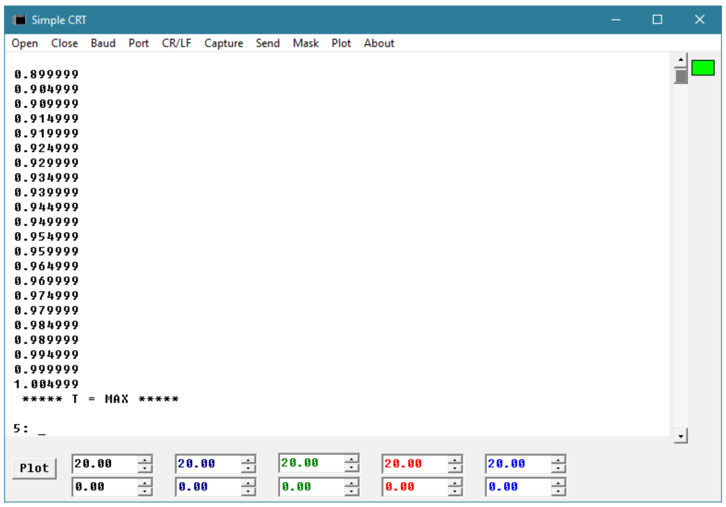
In simpleCRT, select Port then click on your Com number. Next click open and the red box in the upper right corner will turn green if the port opens. Press the Enter key and the Pico will respond with:

***** ??? *****

0:

The response says, I don't understand the command. The 0: tells you the program editor is at line number zero. When you enter program block functions, the line number increases. When you enter commands, the command is executed and the line number remains the same.

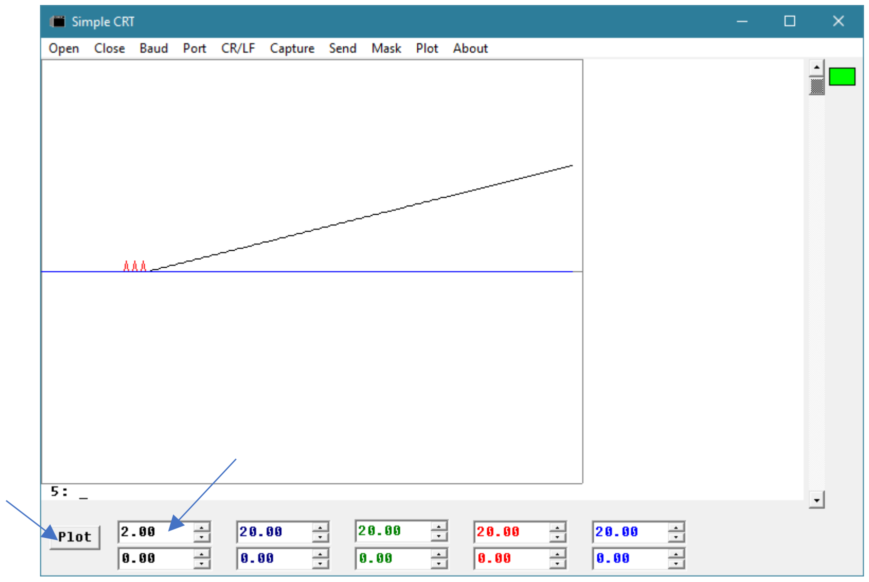
Now, type in the program shown on the previous page. Then use the run command to start it. It will run for one second (max = 1) and print 200 values (max/dt). It actually runs for max + dt seconds due to round off error.



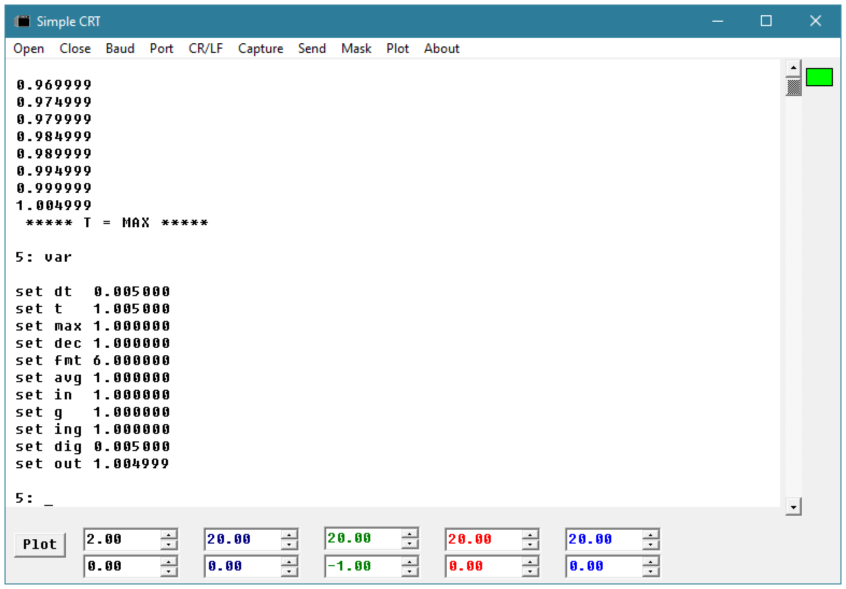
```
0.899999
0.904999
0.909999
0.914999
0.919999
0.924999
0.929999
0.934999
0.939999
0.944999
0.949999
0.954999
0.959999
0.964999
0.969999
0.974999
0.979999
0.984999
0.989999
0.994999
0.999999
1.004999
 ***** T = MAX *****

5: _
```

The solution for our integrator is $g * \int_0^1 k \cdot \delta t + c = \; g * t * k + c;\; t = 1,\;\; g = 1,\;\; k = 1; c = 0$

The output is simply a ramp starting at value c then rising in proportion to time, gain and the input.



Set the first plot gain to 2.00 then click the plot button. The result over time is then plotted as the black trace. The red artifacts are because the plotter looks for numbers in the stored text lines (256 lines max) and it is trying to plot your Block program numbers. Click on the plot to return to the text display.

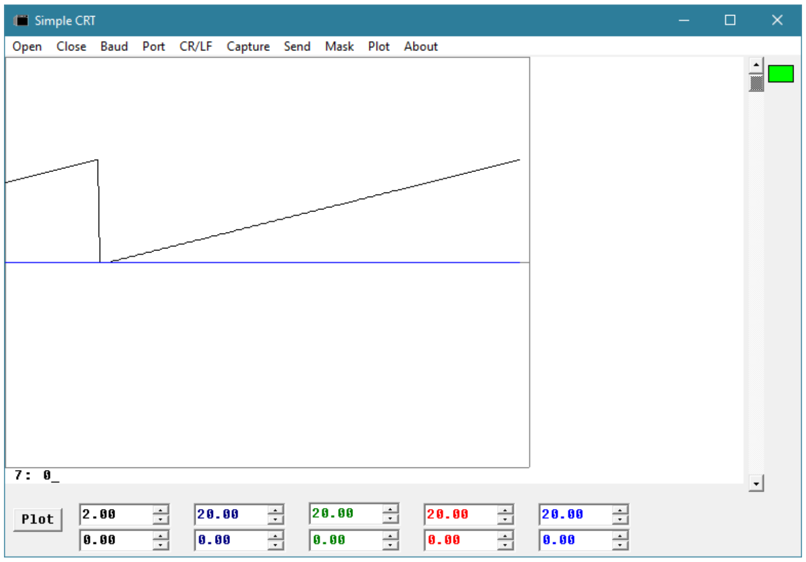


```
0.969999
0.974999
0.979999
0.984999
0.989999
0.994999
0.999999
1.004999
 ***** T = MAX *****

5: var

set dt  0.005000
set t   1.005000
set max 1.000000
set dec 1.000000
set fmt 6.000000
set avg 1.000000
set in  1.000000
set g   1.000000
set ing 1.000000
set dig 0.005000
set out 1.004999

5: _
```

Use the var command to list the current variable values. If you reset time to zero **set t 0** and run the program again, out will ramp from 1.005 to 2.01. The initial condition for our integrator was not reset. So, it starts at the current value.

Enter the following code to reset the initial condition on our integrator. A branch if not zero **bnz t 2** conditional is used to check if t == 0. If it is, the program goes to the next block **rst 0 out** and resets the output. Otherwise, it goes to the second line down **mul in g ing**. The variable **2** (yes variables can have number names) must be set to 2. If you forget, it will default to zero and the program will continuously loop back to the **bnz** conditional block.

```
clr
bnz t 2
rst 0 out
mul in g ing
mul ing dt dig
sum dig out out
prf out
end
set in 1
set out 0.0
set g 1
set dt .005
set max 1
set 2 2
```



```
7: 0_
```

| | | | | |
|---|---|---|---|---|
| 2.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

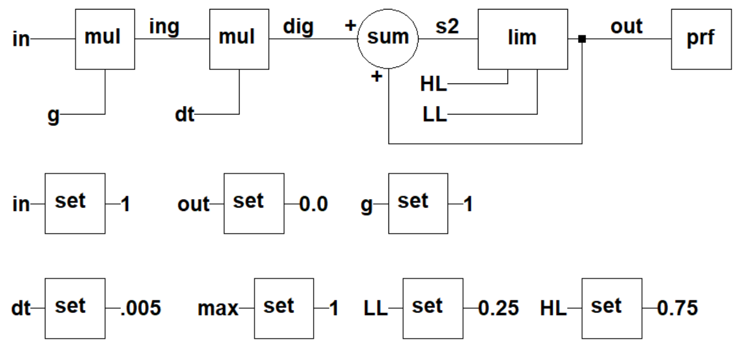
Entering the commands:

```
run

set t 0

run
```

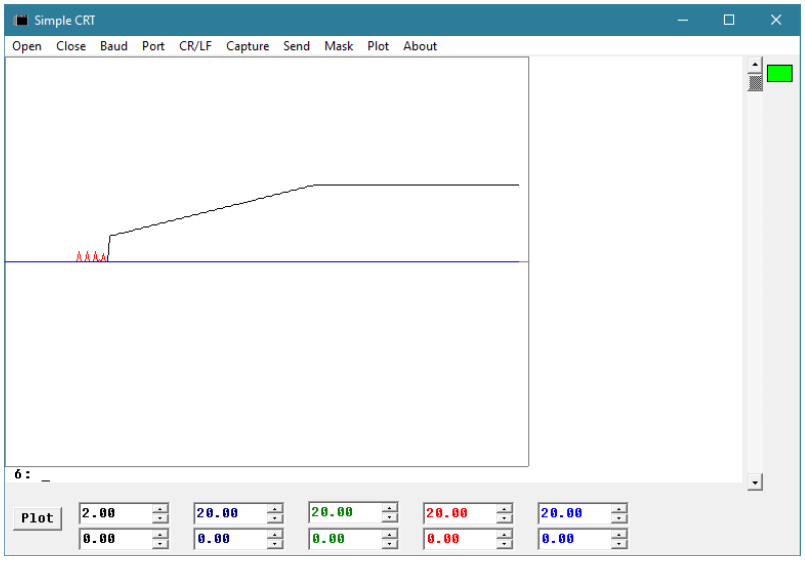
Then plotting the result, shows the integrator initial condition resets. Try adding a conditional branch to reset t if t>max. Hint: **sub t max flg**

Next we will add output limits to our integrator. Limits are used for a variety of purposes. It could be your system can't go to infinity. You have a PID control system and you need to limit the maximum influence of the integrator. A typical system might be a water tank with an overflow. It can't fill below zero and it is limited to a maximum fill height.

Adding a limit function after the integrator does not achieve our goal. The output will limit but the integrator will continue to ramp towards and infinite value. Placing the limit function before the positive feedback of the output will stop the integrator from ramping past the limit.



**clr**
**mul in g ing**
**mul ing dt dig**
**sum dig out s2**
**lim s2 HL LL out**
**prf out**
**end**
**set in 1**
**set out 0.0**
**set g 1**
**set dt .005**
**set max 1**
**set LL 0.25**
**set HL 0.75mul in g ing**
**set HL 0.75**



Since the integrator is limited from 0.25 to 0.75, it can't start at the initial condition zero and it is reset to 0.25. It will ramp at the correct slope until it reaches 0.75 where it stops.

Our integrator took four program blocks. The two built in blocks **eul** and **int** can perform integration with gain and limit functionality. The Euler integration block is rectangular and the **int** block uses trapezoidal.

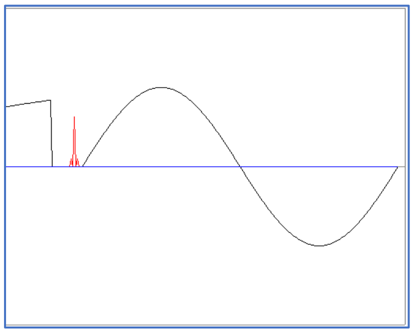
Lets solve this problem $\int \sin(\omega t)\,\delta t = ?$

First we will built a sine wave oscillator. The block **osc** function is locked on time t. All we need is to set a frequency in radians per second, an offset and a magnitude.

out = magnitude * sin(ωt) + offset

Use this program to test the oscillator block:

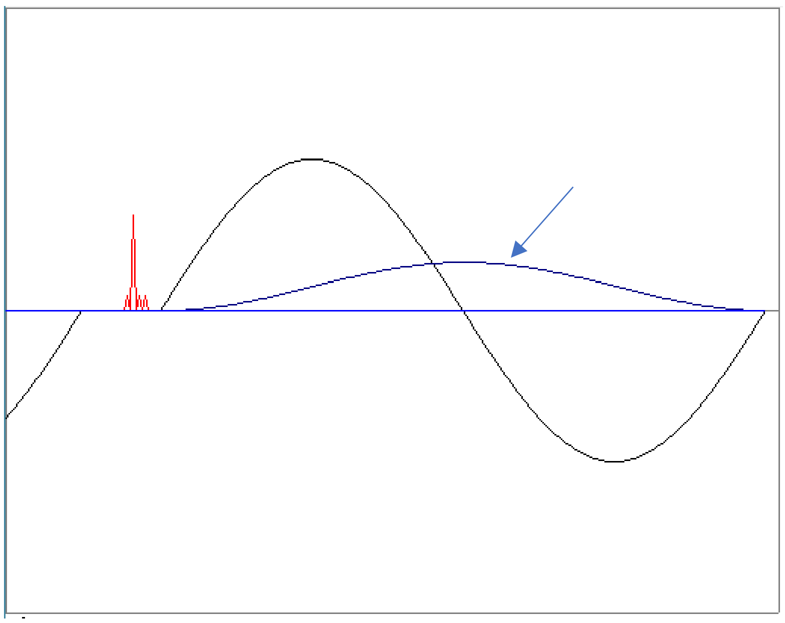
```
clr
osc w os mag out
prt out
end
set dt 0.005
set max 1
set w 6.283
set mag 1
```



It will generate one sine wave cycle.

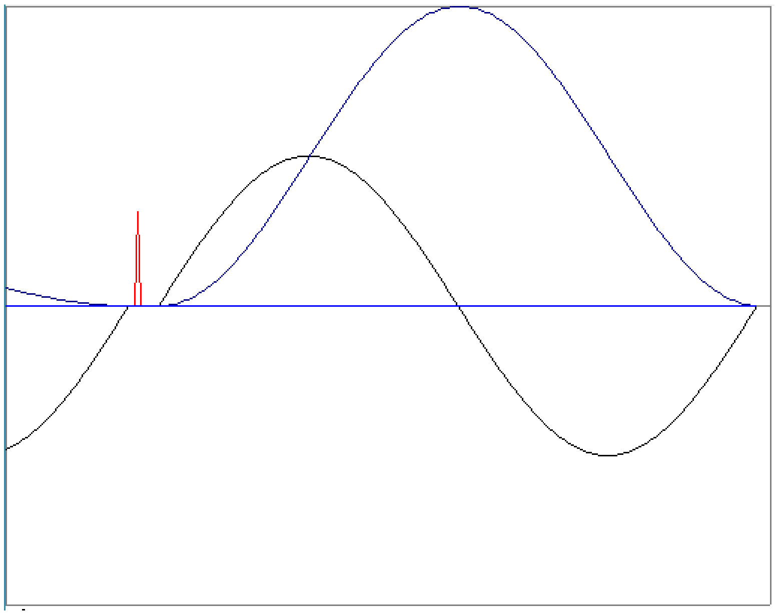
The following block program will integrate the output of the sine wave generator. Its initial condition starts at zero and unity gain is used.

```
clr
osc w os mag out
eul out g lim ans
prt out ans
end
set dt 0.005
set max 1
set w 6.283
set mag 1
set g 1
set ans 0
```



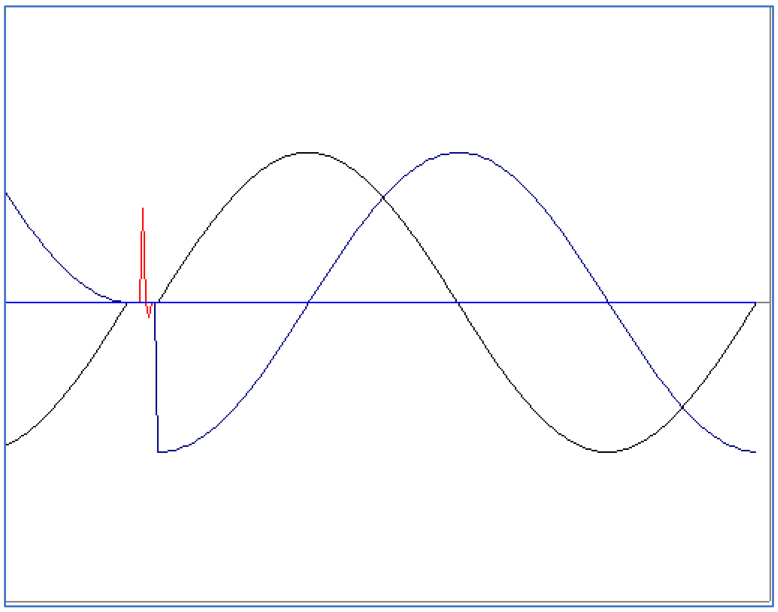
The **ans** part of the plot looks a little small. So, let's add a little gain to the integrator and rerun the program.

```
set t 0
set g 6.283
set ans 0
run
```



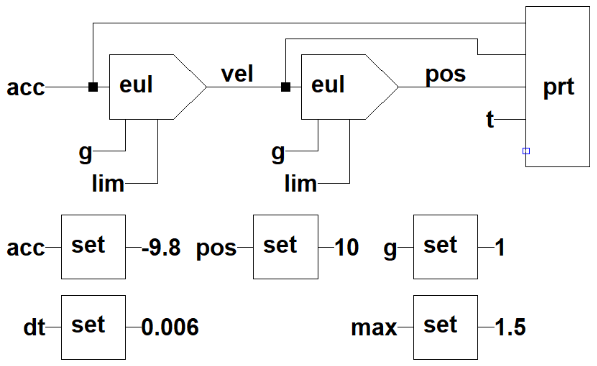
Now the shape of the answer looks a little more familiar. Adding a new initial condition might help.

```
set t 0
set ans -1
run
```



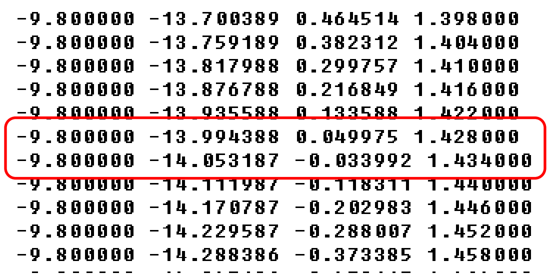
That's better. It looks like $\int \sin(\omega t)\, \delta t = -\frac{1}{\omega}\cos(\omega t)$

Using two integrators, we can find the position of a falling object (in a vacuum) over time. Using the gravity constant of -9.8m/sec^2, integrating once gives velocity over time and integrating a second time gives position over time.

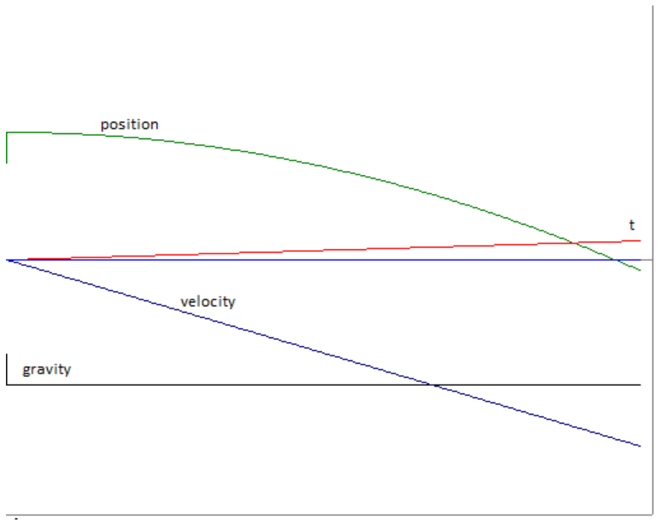


Type in and run the following program to find the velocity and time when an object dropped from 10 meters will hit the ground.

```
clr
eul acc g lim vel
eul vel g lim pos
prt acc vel pos t
set acc -9.8
set pos 10
set g 1
set dt 0.006
set max 1.5
end
```

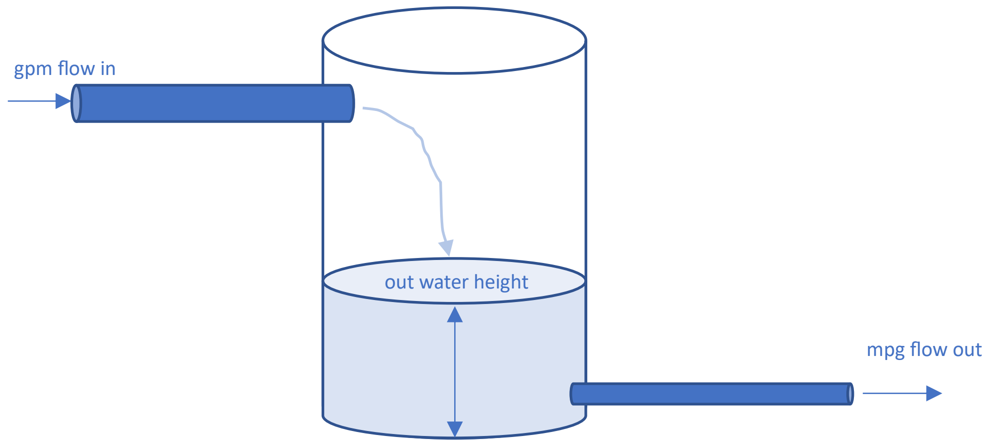


```
 -9.800000 -13.700389 0.464514 1.398000
 -9.800000 -13.759189 0.382312 1.404000
 -9.800000 -13.817988 0.299757 1.410000
 -9.800000 -13.876788 0.216849 1.416000
 -9.800000 -13.935588 0.133588 1.422000
 -9.800000 -13.994388 0.049975 1.428000
 -9.800000 -14.053187 -0.033992 1.434000
 -9.800000 -14.111987 -0.118311 1.440000
 -9.800000 -14.170787 -0.202983 1.446000
 -9.800000 -14.229587 -0.288007 1.452000
 -9.800000 -14.288386 -0.373385 1.458000
```



Right around 1.43 seconds the object hits the ground at 14m/s.

Next, we can used negative feedback to model a tank filling at a constant rate (gallons per minute) while at the same time draining with a laminar flow (water height linear dependency).

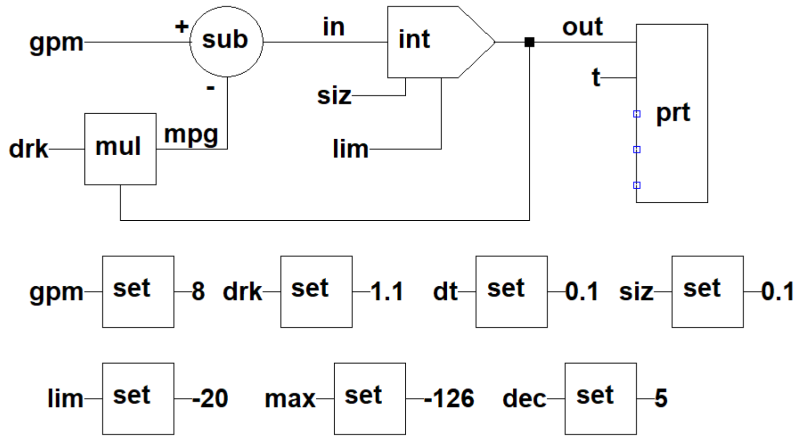


Let:

gpm = 8 gallons per minute in flow
out = tank height in feet
mpg = out * 1.1 gallons per minute out flow

The tank can be represented with an integrator. The integrator output represents the height of water in the tank. Since height is dependent on volume, the gain constant is feet per gallon.

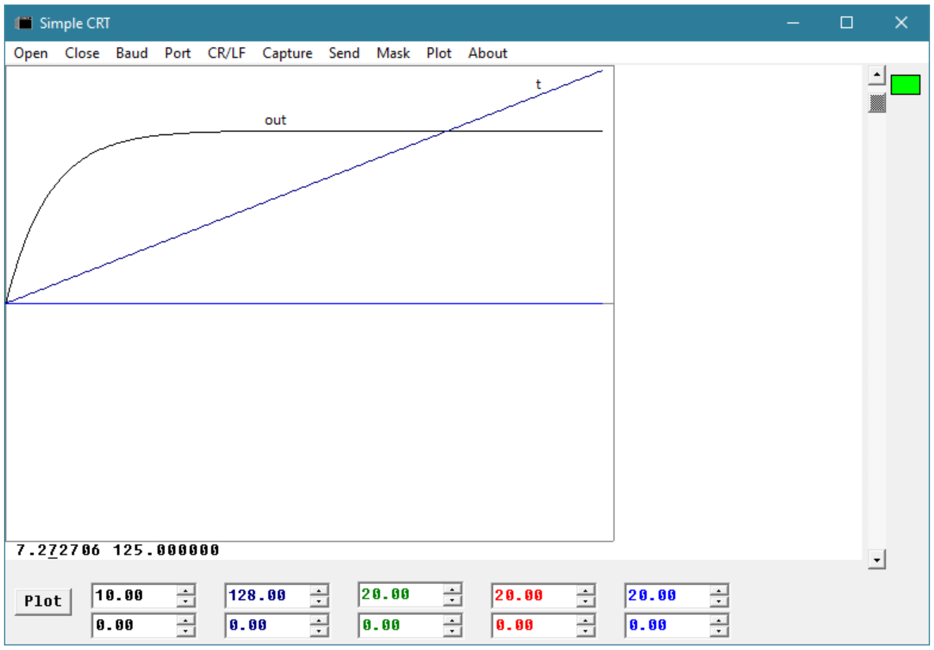
Let: Tank volume = 100 gallons, then for a 10ft high tank: gain = 10ft/100gallons = 0.1



Max is set to a negative value to run as fast as possible. You don’t want to wait 126 seconds for the result. The print is decimated by five. This fits the plot within 256 values.

Here is the tank program to type in and run:

```
clr
mul drk out mpg
sub gpm mpg in
int in siz lim out
prt out t
end
set gpm 8
set drk 1.1
set dt 0.1
set siz 0.1
set lim -100
set max -126
set dec 5
```



```
7.272706 125.000000
```

The tank height reaches equilibrium at 7.27ft.

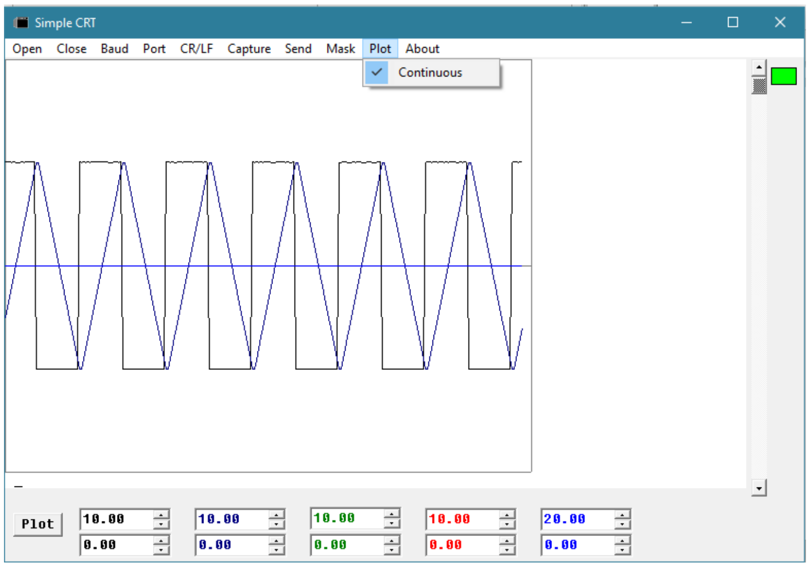
For our last experiment, we will use the analog section. This requires the Pico Block board. A square wave generator will produce an 8Hz plus and minus 5V signal and send it to DAC1. A jumper wire will connect the output of DAC1 to the input of ADC0. The ADC reading is fed into an integrator. Since the voltage is plus then minus, the integrator will ramp up then down in a triangle waveform. The triangle wave is fed to DAC0. The output of DAC0 connects to ADC1.

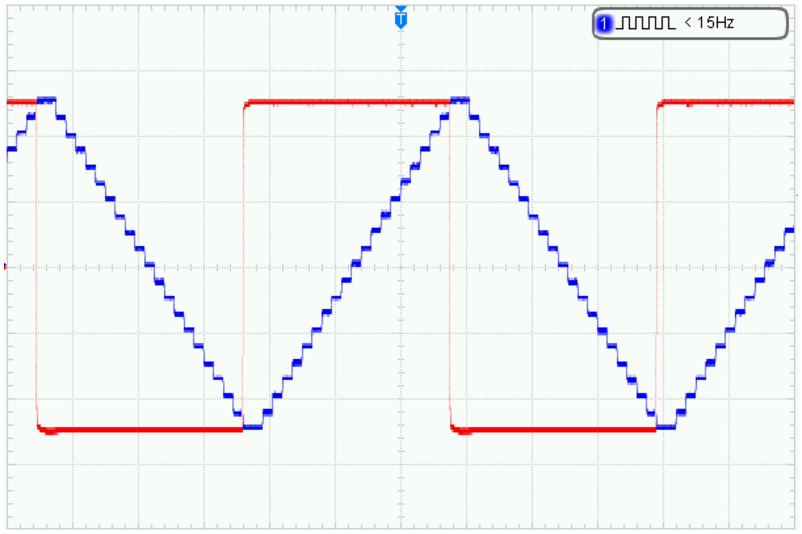
The voltages read on ADC0 and ADC1 are printed. This program runs in real-time. To plot the signals, enable Continuous Plot in simpleCRT.

The step value dt is set to 3ms to allow printing. If you have an oscilloscope, you can change the square wave time and dt to run faster but remove the **prt** block. The integrator gain should be set to 2/tw.

Here is the code to run the triangle wave generator. It is left as an exercise for you to change the code to run without the ADCs and DACs.

```
clr
adc ch0 in0
adc ch1 in1
dac sw ch1
dac tri ch0
swg tw sw
eul in0 g lim tri
prt in0 in1
end
set tw 0.06
set dt .003
set max 10000
set sw 5
set g 33.33
set ch1 1
set tri -5
set lim 5
```





The stepped DAC output is visible on an o'scope.

Note: Remember to use the **cal** command to zero the ADCs. Failure to do so may result in an offset and clipped triangle wave.