Simulations: Also Get the Excel Demo From Course Home Page!

A "simulation" is an animation for which the details of the animation are *calculated*, as opposed to merely *drawn* or otherwise *approximated*. Simulations can be extremely helpful for an audience, because they allow them to see how multiple pieces move and fit together exactly. So, in some sense, they are the same as a PowerPoint animation. But, when we drew the orbit of the planet earth on our homework, the animations were not numerically correct. On the other hand, it might be accurate to call our animation of the gears a "simulation", because we used our measurements of the gear ratios for the different size gears (or, at the very least, our count of the number of teeth on each gear) to make an animation such that the teeth intersected *properly*.

There are many tools available for making simulations. Some of them are free or "free-ish", like "Phun" (https://phun.en.softonic.com/?ex=DSK-309.4). The Phun simulator lets you draw stuff, including solids, liquids, ropes, strings, and springs, in a gravitational field, and then just let them go to see what happens.

A similar simulator from the old days was called "interactive physics" (https://www.design-simulation.com/ip/). This tool allows you to calculate, plot, and export data from your simulations (like positions and velocities).

But any tool that can simultaneously calculate stuff *and* plot stuff can be used to create *custom* simulations. We own at least two of these tools already: Excel and Mathematica. We'll start by creating some simulations in Excel. Later, we'll also be using Mathematica. Neither Excel nor Mathematica is free; but, the college has already paid for you to use both while you're a student here.

Graphics In Excel: see Excel Tab: 0-Scatter Plots

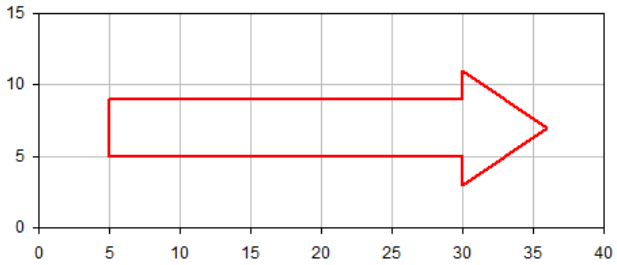
x	***y***
5	5
30	5
30	3
36	7
30	11
30	9
5	9
5	5

Before we create a simulation, we need to be able to make a *drawing* in Excel. The techniques we use here are also used in Mathematica (and even in LabVIEW!). Drawings in Excel take place inside a **scatter plot**. Consider the following data set:

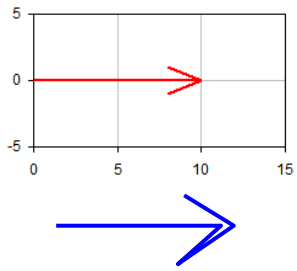
From just looking at these numeric values, it's hard to see how it might relate to graphics or drawing. Let's make a scatter plot. For this plot, we want lines (not smoothed) and no markers. Right away we see that this data is a *drawing of an arrow pointing to the right*. Next, delete any captions or legends. Now, we need the on-screen aspect ratio to be 1:1, which it NEVER is by default. One way to help with this is to temporarily add major axis gridlines. Better yet, get a ruler and measure on your screen. I made my gridlines go in steps of +5 for each axis(Axis Options/Units/Major).

When I measure the default scales, the horizontal range (from 1 to 10) occupies 1.15 inches on my office screen. This is 0.115 screen inches per unit. The vertical range from 1 to 15 occupies 2.73 inches, which is 0.182 screen inches per unit. The ratios need to be the same! So, I want to grab a corner of the plot and stretch it until the ranges are equal per scale unit (or, the major axis gridlines form perfect squares instead of rectangles).

It doesn't matter that this is an "arrow"; you can represent *any object* as a collection of lines, and a collection of *lines* is really just a collection of connected endpoints or *dots*, like this original data here. Note that for "closed" figures like this arrow, the last coordinate must match the first. We'll do some *simpler* "line" arrows in our homework assignments, so let's examine that:



x	*y*
0	0
10	0
8	-1
10	0
8	1



All objects must be "traceable" without picking up your pencil, so this particular arrow retraces over the lower arrow head before drawing the upper arrow head, which I tried to exaggerate here in blue. So, the (10, 0) data point appears twice, for a total of 5 coordinates for this arrow.

Modifying Drawing Objects with Formulas: See Excel Tab: 1-"Live" Arrow

We often desire to modify "base" graphics of this type for various reasons. If this simple arrow represents "force" in some simulated animation, it's possible that the magnitude, position, or the angle of the force are changing with time. Sadly, whenever we change the size of a drawing in a plot, Excel will *automatically* ruin the plot scales unless you tell it not to. Thanks, Microsoft Office Developers! So, let's fix that first. Let's force the plot to ALWAYS range from –20 to +20 for both the *x* and *y* directions. Similarly, retype the "5" for each major tick spacing. Then stretch it until the grids look like real squares, not rectangles. Then delete the axis numbers altogether.

The current size of the arrow is "10", and the current direction is "0 degrees", meaning it points to the right. But perhaps later, we'll want a longer or shorter arrow, or an arrow that aims in a different direction. Let's work on that.

Here, the green cells will eventually represent user choices. The orange stuff indicate stuff that we're thinking we might plot. Blue text indicates stuff that we're adding or changing right now. Let's start by letting the user have a *cell* for the length "L" of the arrow, a cell for the angle, and two cells for the starting position.

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
1	L:	10		x orig	y orig				
2	Angle:	0	"deg"	0	0				
3	x	0		=B1	0				
4	y	0		=B1-2	-1				
5				=B1	0				
6	Angle:	=B2*pi()/180	"rad"	=B1-2	1				
7									

Now, the user can change the length in cell B2. Note that for this simple arrow, the size of the arrowhead is *not* proportional to the size; it's fixed. So if you choose a small arrow magnitude, it will look pretty stupid. So, next let's make the arrowhead *proportional* to the length of the arrow. We'll do this by replacing the hard-coded "1"s and "2"in cells D4, D6, E4, and E6 to refer to the length of the arrow.

Next, let's think about allowing the user to rotate the arrow. We do this with trig. There are many ways to think about it. It's based on a rotation matrix, but here, I'm just entering the formulae by hand in the first row. Once the top row is entered, drag them down over the whole orange region.

	A	B	C	D	E	F	G
1	L:	10		x orig	y orig	x rotated	y rotated
2	Angle:	0	"deg"	0	0	=D2*B$8-E2*B$7	=E2*B$8+D2*B$7
3	x	0		=B1	0	(drag)	(drag)
4	y	0		=0.8*B1	-B1/10	(drag)	(drag)
5				=B1	0	(drag)	(drag)
6	Angle:	=B2*pi()/ 180	"rad"	=0.8*B1	B1/10	(drag)	(drag)
7	sine:	=sin(B6)					
8	cosine:	=cos(B6)					

To see the change, you'll need to adjust the plot to use the new columns. Right-click on the plot, then choose "select data", then "edit", then DELETE the "series x" data, and replace it by dragging down column F. Do the same thing for the "series y" data, replacing it with column G.

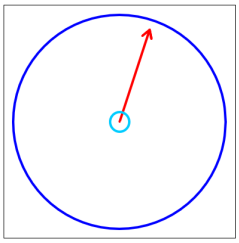
Finally, let's see how this arrow could be moved to a new position within the plot window. We just add the offsets in cells B3 and B4 to the already rotated image:

	…	D	E	F	G	H	I
1	…	x orig	y orig	x rotated	y rotated	x final	y final
2	…	…	…	=D2*B$8-E2*B$7	=E2*B$8+D2*B$7	=F2+B$3	=G2+B$4
3	…	…	…	=D3*B$8-E3*B$7	=E3*B$8+D3*B$7	(drag)	(drag)
4	…	…	…	=D4*B$8-E4*B$7	=E4*B$8+D4*B$7	(drag)	(drag)
5	…	…	…	=D5*B$8-E5*B$7	=E5*B$8+D5*B$7	(drag)	(drag)
6	…	…	…	=D6*B$8-E6*B$7	=E6*B$8+D6*B$7	(drag)	(drag)
7	…						
8	…						

Again, after typing the first row, drag the upper row down to fill in all the rest of the green rows. As before, we need to reselect the data being drawn to use columns H and I.

To see the impact of all of this, type new numbers into any of the cells B1 through B4 (the "green" user controls for this drawing).

Making Simulations in Excel II: Simple "clock"
See Excel Tab: 2-Clock+Slider



Let's make a "simulation" of the minute hand of a clock. There is very little physics here… it's just to get used to the interface in Excel. This first version will allow users to control the length of the arrow, the size of the arrow head, and the "time" spacing between the frames of the animation. Also, the user will have a slider control to choose which frame of the "video" to see right now. Later, we'll learn how to add in a "start" button so that the frames will all play in the proper sequence at some rate.

As with most simulations, we'll need to mentally divide the worksheet into several regions:

1. **User controls**. Usually these are in the top left corner of the sheet! We'll be adding in the slider controls as the very last step.
   Here, I'm using cells B2 through D17.
2. **Global calculations**. These can be anywhere. Often, we hide them from the user (by placing them in columns far off to the right). Other times, we show the user only the "main" computations. So, for example, if a user was inputting a frequency of a photon, we might choose to display the wavelength for them as a reference.
   Here, I'm using cells B19 through D24.
3. **Frame-specific calculations.** These are usually hidden to the user by placing them in columns "way off to the right". But, I'm keeping them close today so we can see everything at once, as developers. Also, note that the height of the columns we use sets an absolute limit on the number of frames my "video" can possess. Here, I'm using 121 frames, numbered 0 through 120.
   Here, I'm using cells F3 through J124.
4. **Scatter-plot (x, y) data for each series to be plotted**. As seen above, my plot will have three series: a small inner circle, a larger outer circle, and the minute hand itself (an arrow).
   Here, I'm using cells L7 through T43 (orange).
5. **The plot itself.** Obviously, this needs to be very close to region 1, so our user can see it!

Note that modern versions of Excel misbehave when the zoom level is not at exactly 100%. So, try to keep at that Zoom level during your development, especially when you choose your plot size.

One of the "global calculations" is called "current index". It's bright yellow. It's just a number from 0 to 120 that will be used to select *which* frame number to display right now. Although in some sense, this is a user control, this cell will later be controlled by the slider. That's why it's not in the "user controls" section. Even with the sliders present, there's no problem if the user just types a number in this cell (as long as it's in the range 0 to 120!)

Also in the global calculations section is a number for "omega". I've just put in a value of $2\pi/60$ radians per second. This is the correct angular speed for a clock's minute hand. I'll come back to the remaining "global calculations" later.

The frame-specific calculations are the main thing. You'll see I have something called "index", which might also be called "video frame number". These are just integers from 0 to 120. The next column is "time", in seconds. It's the frame number times the "time step size" chosen by the user. So, if the chosen time step is 0.5 seconds, then each frame will represent times that increment by 0.5 seconds from each other. So, the final frame has a time of $0.5\times120 = 60$ seconds, and we'd expect the minute hand to make 1 revolution. However, if the user selects time steps of 1 second per frame, then the 120 frame video will have a final time of 120 seconds, meaning the minute hand will have rotated twice. Also, there's a column for θ for this arrow, which is just ωt. That's all the physics for this first simulation!

The next two columns calculate the (x, y) coordinates of the "point" of the minute hand arrow. They are just $r\cos\theta$ and $r\sin\theta$, as you'd expect. Other simulations would have more frame-specific stuff to calculate, but this is enough for this sim.

Next, let's finish up the "global calculations". We have data for many frames, and now we want to extract the (x, y, θ) info of the arrow *for only the frame selected in bright yellow*. To do this, we'll use a cool Excel function called VLOOKUP ("V" is for "vertical").

The VLOOKUP function needs 4 parameters. The first one is the frame number we care about (i.e., the yellow cell). So it's obviously just C20. The next is a 2D table, arranged in columns. The first (leftmost) column must be frame numbers. The rest of the columns will be the data we extract for the chosen frame. My table is F4 through J124 (**index**, **time**, **theta**, **x,** and **y**). In VLOOKUP, the next parameter is the column we want to extract from the table, and the last parameter doesn't matter today (just use 1). So, using column 3 means "extract theta", and using column 5 means "extract y". Here, I used VLOOKUP three times: to get the values of x, y, and θ for the minute hand at the chosen frame.

Next, we look at the drawing data. For the two circles, I made a column of angles in 10 degree increments, going from 0 to 360 (so, 37 values). The last is the same as the first so I can "close" the figure. If I had chosen increments of 45 degrees (8 points + 1 more to close it), then I would get an octagon instead of a circle. For the arrow, I did all the same stuff we did in the earlier sheet "live arrow", but without any side-to-side or up-and-down offsets. Also, I put all the math into a single column instead of spreading it out conceptually over many columns.

At this point, our simulation works! To see new frames, the user has to type numbers into the bright yellow cell C20. Try it yourself!

Now let's add the sliders to make it easier to update the numbers in cell C20.
Turn on the "developer" tab. In that tab, turn on "Design Mode" (if you're using a PC). Go to "insert".

If you are a PC user, choose the "Scroll Bar" in the "ActiveX" submenu. Then drag a (mostly horizontal) region somewhere in your sheet to draw & add the scrollbar. Then right-click it, activate "properties", and make the following adjustments:

Linked Cell = C20; Max = 120; Min = 0 (probably already true!)

You have to turn off design mode to *use* the scrollbar, and the scroll bar may not be actively selected. You can hold down the left or right endpoints to get a smooth stream of updates!

If you are a Mac user, instead get the "Scroll bar" from the "Form Controls" submenu (PC users can also do this, but the result is inferior, so why bother?) Right click, and "format control". In the "control" tab, set the min/max/linked cell in the same way as above. Sadly, this scroll bar will not actually update our animation until you let go of it! So, you have to "fast click" on the endpoints to make a smooth animation.

Adding a "Start" Button to the Clock Simulation: See Excel Tab: 3-Clock+Button

Now, instead of "sliding" to run the simulation, I want to add a "play video" kind of button.

Almost everything here is the same as the previous worksheet. However, I got rid of the scrollbars, and added a new user input for "animation speed". Also, I added a new *button*. It came from Developer/Insert/Form Controls/Button. Immediately, you'll get a window asking you to create a "macro" (i.e., a Visual Basic program) to go with this button. You can change the name of the new macro to "StartButton" if you want, but in any case, click on "new". This takes you into a Visual Basic program editor. You can freely move back and forth between regular Excel and the VB editor. Saving the Excel document also saves the VB programs. When you eventually get back into regular Excel, right-click the button to edit the button text, and change the name to "Start Clock", or something like that.

But for now, let's put in some VB code! If you've accidentally closed the VB window, you can get it back from the Developer Tab/View Code. The text of my code is pasted on the next page, and it's probably a little fancier than it needs to be. You can copy and paste (but of course, it's already in there anyway). Stuff in green is just comments.

Here's a summary:

1. Get the user's number for speed, and make sure it's between 10 and 200. If it's not, overwrite the user's value!
2. Starting at frame 0, count through all the frames using "index" as the current frame number. Each time we look at a frame, overwrite cell C9 (bright yellow) with the newest frame number.
3. Force the plot to update with the current value.
4. Wait for a short delay to try to control the speed. This method has limits, because stuff that's always happening takes time, too, so there's a maximum possible speed.
5. Update the frame variable "index" to look at the next frame.
6. When done, reset the current frame number in cell C9 to zero.

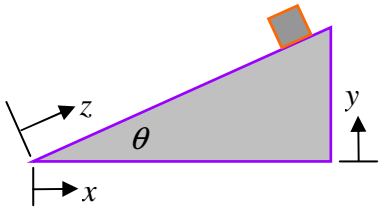
```
Sub Clock_Button_Click()
Dim index As Integer
Dim time0, time1, speed, dt As Double
    index = 0
    Range("C6").Select
    speed = ActiveCell.Value       ' desired speed is in cell C6
    If speed > 200 Then            ' set max allowed speed to 200
        speed = 200
        ActiveCell.Value = 200
    End If
    If speed < 10 Then             ' set min allowed speed to 10
        speed = 10
        ActiveCell.Value = 10
    End If
    dt = 1 / speed                     ' set time delay between frames
    time0 = Timer                      ' get start value for delay
    While (index < 121)                ' main sheet only has 121 frames!
       ActiveSheet.ChartObjects("Chart 1").Activate
       ActiveSheet.ChartObjects("Chart 1").Chart.Refresh  'continuous updates!
       Range("C9").Select              ' location of current frame number...
       ActiveCell.FormulaR1C1 = index  'update frame number now!
       DoEvents                        ' needed for Macs, optional for PCs
       time1 = Timer                   'use computer clock to get time "now"
       While (time1 < time0 + dt)      ' delay for amount of time "dt"
          time1 = Timer
       Wend
       time0 = Timer                   ' reset start time for next frame
       index = index + 1               'move on to next frame...
    Wend
    Range("C9").Select                 'at end of sim, reset index to zero
    ActiveCell.FormulaR1C1 = 0
    Range("A1").Select
End Sub
```

There's a possible hidden issue in this code. Twice, it refers to "Chart 1". However, when you make a worksheet, it might have several charts, or even deleted charts. So, perhaps *your* chart is really "Chart 7". Here's how to find out… once you have a plot made that you want to refer to, go to the Developer tab. Off to the left, click on "Record Macro". The name of the macro you create doesn't matter, so just click "ok". Now, Click on your plot, then click in any cell. Then, in the top left corner, "Stop Recording" (looks like this: ☐). Then click on "Macros", choose the same name that you just recorded, and then click "edit". You'll see a very short VB program that tells you the name of this plot. Make sure you use this actual chart name in the code for your button. You can delete this little macro afterwards, if you want. In fact, using this "Record Macro" tool is a great way to learn how to code VB programs to do Excel stuff… it's how I learned almost everything!

Making Simulations in Excel II: Block on an Inclined Plane: See Tab: 4-Incline+Button

Let's simulate something more physicsy. The concepts are all very similar to those we saw for the clock. We will make a simulation of a block sliding down a hill. During the simulation, we'll *show* both the hill and the block (2 series). This figure shows the coordinate system I will use. The box can start at any position on the hill, with any initial speed.

We'll have to make some decisions about the time step between frames, the maximum duration of the simulation, permitted values for geometry, and so on. In this example, I didn't add very much error checking at all… if the user selects stupid parameter values, this code doesn't do much to try to correct them!

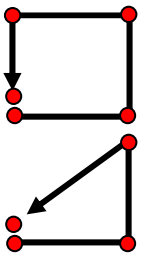


Region 1: User Controls: I've chosen to have the user inputs be θ, v_0, h_0, w (the box size), g, and of course, a start button. I also added a "reset" button. Sometimes, when the user makes value changes, the plot won't update until either button is pressed.

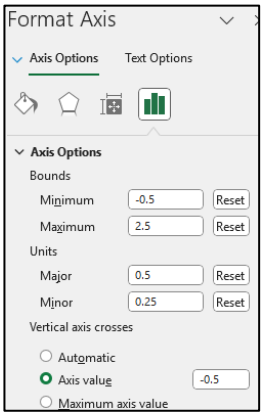
Region 2: Global Calculations: this is cells P11 through R25. One "new" thing to notice is how I did "error checking" for θ. I wanted to restrict the hill angle to be less than 45°. So, cell Q12 finds the minimum of the user selected angle and 45. All subsequent calculations that use the angle will then use *this* cell Q12 instead of the "actual" value requested by the human. I could do more error checking, too, but I didn't bother to add other restrictions in this demo.

Also, you can change the "hill width" of 2 m if you want, but you'll also need to rescale the plot if you do. Currently, the horizontal plot width ranges from -0.5 m to +2.5 m, and the floor of the hill itself ranges from 0 m to 2 m. So, if you change the hill width to 3 m, then part of it will be off the right-hand edge of the plot. Also, there's a wee bit of incomplete error checking that makes sure that the box can't start out higher than any point on the hill. You should examine the formula bar for all the yellow cells in this section to see what they are doing. As with all learning, do not bother to try to figure out *why* these cells are doing this until **after** you are 100% clear on *what* they are doing.

Region 3: Frame Data: This is in columns I/J/K. The physics for the "z" value will be discussed later. For the time steps, I made the following decisions: I chose to create only 101 frames, ranging from 0 to 100. Then, I used physics to calculate the time when the box first reaches the bottom of the hill. I called this t_{bottom}. Then, I chose my time step to be $\Delta t = (t_{bottom} /100)$. So, the simulation will always end whenever the box reaches the bottom of the hill.



Region 4: Plot Data: These are pretty short, because the hill is just 3 lines connecting 4 dots, and the box is 4 lines connecting 5 dots. I first find the box coordinates as if it was on the ground and flat, and then I use the same techniques we learned earlier to angle the box and put it at the "chosen" position on the hill.



Region 5: The Plot: The plot possesses two series. Also, you need to make sure that the axes ranges are selected by *you*, the developer, and are not "automatic". If you don't, then the plot will keep rescaling itself during the simulation, which is like being on drugs. This is fixed by just changing **each** of the numbers by hand! Here, I chose -0.5 m to +0.25 m for each axis (as mentioned earlier), and I added light gray gridlines with a spacing of 0.5 m. I also moved the axis numbers down to -0.5 m, as seen in the last box of this screen shot.

Box on a Hill: The Physics

If it doesn't have physics, it's not a "simulation"! Note again that all trig calculations like $\sin\theta$ require radians. Hopefully, we already recall that the acceleration of the block is $-g\sin\theta$ along the plane of the hill "z" ["Equation 1"]. Consequently, the position of the block as a function of time is:

$$z = z_0 + v_0 t - \frac{1}{2} g \sin\theta \cdot t^2 \qquad \text{"EQ 4"}$$

In the calculations, I also use these: $v = v_0 + a\,t$ "EQ 2"

$v^2 = v_0^2 + 2a(x - x_0)$ "EQ 3"

The Drawing

I started the drawing by making a scatter plot for the 4 coordinates of the hill, as described earlier. Then, I added 5 pairs of coordinates for a non-inclined box located at the origin. Then I used trig in two ways to make the final 5 pairs of (x, y) coordinates for the box: first, I calculated $x_{\text{offset}} = z\cos(\theta)$ and $y_{\text{offset}} = z\sin(\theta)$ from the z coordinate found in cell C38. then, I used the necessary trig to compute the corner coordinate of the box when rotated by the angle θ:

$$x_{\text{rotated}} = (x_{\text{unrotated}})\cos(\theta) - (y_{\text{unrotated}})\sin(\theta)$$
$$y_{\text{rotated}} = (x_{\text{unrotated}})\sin(\theta) + (y_{\text{unrotated}})\cos(\theta)$$

The new corners of the box incorporating both trigonometric effects are in cells N4:O8.

Now, the user can specify any frame and see it "live" on the scatter plot. However, we have to make sure that the major axes spacings are uniform and to NOT allow Excel to recalculate them whenever the user chooses a new angle. I dragged the final corners of the plot until the aspect ratio was correct. It might rescale on a different monitor, so you might have to re-adjust.

Turning off "Design Mode"

You can close the Visual Basic/Macro editor at any time. The code you entered is saved when you save the entire workbook. However, even when you close the Macro editor, Excel might still be in "Designer" mode. On the Developer Tab of the ribbon, look at this icon. If it's highlighted as seen here, click it once so that it's no longer green. When it's green, the workbook is in programmer mode, but when it's white, it's in user mode. As far as I can tell, in modern Excel, you don't need design mode unless you use the "ActiveX" versions of the controls.