Simulations

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). That 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”. This tool allowed 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. But, in the next week or two, we’ll be using Mathematica. Mathematica is NOT free. However, SUNY Geneseo has ALREADY paid Wolfram Research, Inc. $50 for every Geneseo student so that you can use Mathematica if you want. Therefore, you can’t just download it like the other software we’ve used; if you don’t follow the instructions, you won’t be able to use Mathematica without paying your own $50. Here is where to find the instructions to “claim” your already-paid-for right to use Mathematica as a Geneseo student:

https://wiki.geneseo.edu/display/cit/Mathematica+Installation+and+Licensing+Instructions

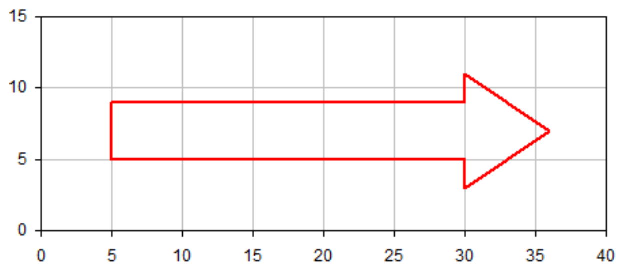
So, before you come to class next week, you need to download Mathematica. If you are a junior or senior, you already did this long ago.

Graphics In Excel

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:

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

When I make a standard scatter plot of this data, it's hard to see how it might relate to graphics or drawing. Let's make some adjustments. First, we need the on-screen scale to be 1:1, which it NEVER is by default. So, look at the axis labels. Better yet, get a ruler and measure on your screen.

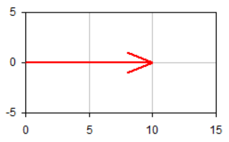


By default, when I measure the scales, the horizontal range from 1 to 10 occupies 1.15 inches on my office screen, but the vertical range from 1 to 10 occupies 1.82 inches. So, I want to stretch the plot until the ranges are equal per scale unit. Before doing this, I deleted the caption, then I dragged the top of the plot down.

It's still hard to see how this connects to "graphics". However, if I now double click any dot, I can change the "marker" to "none", and the line to "solid". It's now clear that this data represents a drawing of an arrow pointing to the right.

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 collected of connected endpoints of dots, like this original data here. We'll do some *simpler* arrows in our homework assignments, so let's examine that:

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



We often desire to modify graphics of this type for various reasons. If this second arrow represents "force" in some simulation, 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. Here, the orange cells will eventually represent user choices. Let's start by letting the user have a way to change the length "L" of the arrow:

	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									

We'll **plot** the green stuff. 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, 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 green 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. 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.

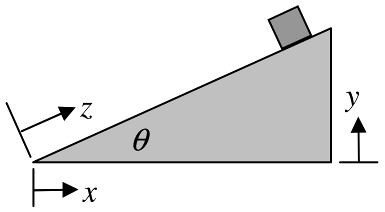
To see the impact of all of this, type new numbers into any of the cells B1 through B4 (the user controls for this drawing).

Making Simulations in Excel

Now that we can draw, let's simulate! The basic idea of a simulation is to use some "physics equations" to compute a set of dependent variables as a function of one or more independent variables or properties. In the preceding calculations, the independent properties were length, angle, *x* position, and *y* position. There was no real dependent variable in the preceding work; it was just a drawing of an arrow rather than a simulation.

In many simulations, the main independent variable will be time. To see how this works, let's make a simulation of a block sliding down a hill. During the simulation, we'll show the hill and the block, both as functions of time. Without repeating the derivation, the acceleration of the block is $-g\sin\theta$ along the plane of the hill "*z*". Consequently, the position of the block as a function of time is:

$$z = z_0 + v_0 t - \frac{1}{2} g \sin\theta \cdot t^2$$



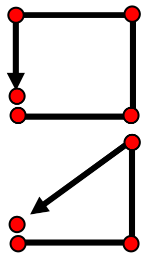
This is a tricky problem because in this formula, *x* is along the plane of the hill, rather than being horizontal.

We'll have to make some decisions about what we want to be able to show (for example, how long does the simulation run?) and what kind of data we'll accept from the user (can I choose a hill that's 3 miles high?). Also, we'll have to organize our interface somewhat so that it's easy to use by a person, and also easy to follow for a programmer (you!).

For this simulation, I feel like I need at least 6 separate "regions" within the spreadsheet itself:

1. A place for users to enter preferences like initial position, hill angle, etc. For now, this should include a way for the user to choose when "now" is. So, for example, if a simulation runs for 3.0 seconds, the user should be able to choose to display the graphics when t = 2.7 seconds (or any other time). Later, we'll find a way to let Excel choose a *sequence* of all the possible times in the simulation automatically. This user data-entry region should be in the upper-left corner of the spreadsheet, and in the best cases, there will be some means of error-checking to prevent the user from entering nonsensical data.
   In the best cases, the rest of the regions are hidden from the user so they aren't distracting, but I have kept them out in the open in this example so you can view them as designers rather than users.
2. A place to calculate commonly needed constants, like sin(θ).
3. A tall set of columns to calculate all interesting results as a function of time. For a generalized motion simulation, we'd need columns called "index", "time", "position", "velocity", and "acceleration". However, in this simulation, I'm just using the first three of these. "Index" is just a column of integers counting from 0 to some larger number. You can think of these as "video frame numbers" for the eventual simulation.

4. A place where we use the chosen "time" from step 1 to pick out a particular data point (or video frame) from step 3.
5. A place where we make columns of (x, y) graphics information for every visible object in the simulation at the chosen time. In this case, we need drawings of a hill and of a box. A box has 4 sides, so as seen in this sketch, we need 5 endpoints to draw it. The hill is a triangle, so it only requires 3 sides and 4 endpoints to draw it.
6. A scatter plot to actually draw all the objects.



Box on a Hill: Programming the Physics

If it doesn't have physics, it's not a "simulation". Here's how I programmed this demo.

The user choices are the angle of the hill (which is restricted to 45°. The restriction allows me to draw the field of view without worrying about whether the hill will poke out of the top of the drawing.), the initial velocity of the block (i.e., it might be moving upwards or downwards at the start of the animation), the initial height (in the y direction), the size of the box, and even "g" to simulate the behavior on other planets. Also, I've restricted the initial height so that if a user enters a number that's too big, it gets reduced so that it's only at the top of the hill.

Note that my users are entering their angle in degrees, but Excel only wants angles in radians, so I have an extra cell (C28) where I store the radians version of the angle for future use.

I've forced the horizontal width of the hill to be exactly 2m, so I don't have to worry about whether the viewing field includes all of the hill. It is possible to calculate and apply a new viewing field on-the-fly as the user makes choices, but I'm not doing that here.

The time discretization is 100 time steps. First, I calculated the amount of time needed for the box to reach the bottom of the hill as a function of the initial conditions. Then, each frame of the resulting animation is spaced by $\Delta t = t_{\text{total}/}100$.

In the "button" version of this demo, I also allow the user to control the playback speed. Sorry, without owning a Mac, I can't figure out how to get the button version to work on a Mac. It seems that on a Mac, the buttons "sort of" work, but at a *much* slower pace. Also, on the Mac, the drawings are not being updated frame-by-frame; instead, the Mac is only drawing the first and last frames.

In the calculations, I use "Equation 1": $x = x_0 + v_0\, t + ½a\, t^2$
"Equation 2": $v = v_0 + a\, t$
"Equation 3": $v^2 = v_0^2 + 2a(x\text{-}x_0)$

Calculating Results

Most of the work is done in the tall columns F, G, and H. You always start with a column called "index" or "frame number", counting from 0 up to the maximum frame number you want to calculate. Here, I'm counting up to frame #100, so I have 101 frames. On the other hand, it's also possible to allow the simulation to compute the necessary number of frames for a simulation on-the-fly, but it's a little harder. In that case, you'd need a number or a rule for determining the appropriate value of Δt.

The next column is "time", and it's just the constant Δt times the frame number.
In this simulation, the next and final column is "z", the position of the block. For each row of this column, I've just copied equation #1 each time, referring to the neighboring value of t in each case.

Picking a Frame

Cell C8 is currently a user input where the user types the number for the frame that they want to see. Valid choices range from 0 to 100, but I'm not doing any error-checking in this version. So, if the user enters an invalid number, Excel will generate errors. Then, in cell C40, I use the always wonderful "VLOOKUP" function. In this case, the VLOOKUP function takes the desired frame number, then searches through the first column of the tall array of data in cells F4 through H104 until it finds this frame number in the first column. Once found, VLOOKUP then looks across that row to the *second* column to discover the relevant time that is associated with that frame. Similarly, I use VLOOKUP to get the z location of the block for that frame from the third column.

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.

Animating a Simulation: Scroll Bars

There are four ways to control the animation of simulations using Excel. Each is a way for Excel to automatically type in frame numbers into cell C8. They all require you to turn on the developer tab. Also, they all require you to enable "macros" when you open the notebook. Also, once they're added, you have to save as ".xlsm" format. The first two are kinds of "scroll bars", and the second two are kinds of “command buttons”. Also, each tool has two versions: a “forms” version, and an “ActiveX” version. The “ActiveX” versions are nicer, but won’t work on a Mac under any circumstances. The main difference is for “scroll bars”, which are sliders. For an ActiveX slider, updates happen to the spreadsheet continuously while the user drags them. However, for “Form Control” versions that you have to use on a Mac, the user has to let go of the slider before they can actually see the impact of their choices. However, I’ve heard that the Form Controls actually work better on Macs than on PCs.

To add a scroll bar, we have to turn on the Developer Tab of the ribbon. Find a blankish part of the ribbon, then right-click and choose “Customize the Ribbon”. Then, “choose commands from” “Main Tabs”, then highlight the “Developer Tab” and add it to your ribbon using the arrow in the center of the panel. Finalize this by clicking the “OK” button at the bottom.

Now, we can use the Developer Tab. Click on the “insert” tool to get a submenu of choices, then pick the “scroll bar” from the “forms” submenu. This gives a cursor to draw a rectangle somewhere on your spreadsheet. Draw a rectangle that is a skinny horizontal bar. The scrollbar isn’t actually in any cell, it’s just laying on top of them.

Now right-click the scroll bar and go to “format control”. There are 4 properties that matter. Since this scrollbar will be used instead of the user typing in numbers for “index” in cell C8, I want to set the following:

Min = 0
Max = 100
Increment = 1
Linked Cell = C8

These values mean that using the scroll bar, the user can select a number from 0 to 100 inclusive, and that value will be immediately copied into cell C8. If you want the user to only be able to select values like 5, 10, 15, 20, etc. , then you’d set the increment to 5 instead of 1.

Animating a Simulation: Command Buttons

Instead of a scroll bar, we programmers can add in a “command” button. The idea is that when the user presses the button, Excel will progressively change the “frame number” value in cell C8 from 0 to 100 with a small time delay between each frame, all automatically. Here, I’ve added a second button called “reset” which just chooses frame 0 and then stops right away. To get a command button, you follow the same method as the scroll bars.

If you right-click the button, you can edit the text you see on the button face.
Each command button runs a subroutine written in Visual BASIC.
If you right-click, and go to “assign macro”, you’ll see the name of the VBASIC program associated with this button. Right now, it looks like “… button3_click”. If you now choose the “edit” button, you’ll be able to create, see, or edit the code for this VBASIC program, called a “macro”.

In this macro editor, here is the code I entered. Note that the first and last lines are already pre-entered for you by Excel, so you don't need to type them or change them. I put my comments about this code in green:

```
Sub Button3_Click()
Dim index As Integer                 'I 'll need a variable to remember what frame I'm on
Dim time0, time1, dt As Double        'used for controlling the playback speed

   index = 0                         'I want to start at frame zero
   time0 = Timer                     'what time is it now?
   Range("C9").Select                'move the cursor to this cell of the spreadsheet
   dt = 1 / ActiveCell.Value         'the time delay is the inverse of the speed
   While (index < 101)               'start to do everything else until we've finished frame 100
     Range("C8").Select              'move the cursor to this cell of the spreadsheet
     ActiveCell.FormulaR1C1 = index      'have excel type this number into the active cell
     Range("C9").Select               'move the cursor to this cell, "completing" the data entry
     time1 = Timer                     'see what time it is now…
     While (time1 < time0 + dt)   'wait until the desired amount of time has passed
       time1 = Timer
     Wend                             ' "End this While Loop": the time delay is over
     time0 = time1                     'the next frame starts when this one ends…
     index = index + 1                'get ready to do the next frame
   Application.ScreenUpdating = False  'does nothing. I hoped it would make Macs work
   ActiveSheet.ChartObjects("Chart 1").Activate     'select the plot
   ActiveSheet.ChartObjects("Chart 1").Chart.Refresh    'update the plot
   Application.ScreenUpdating = True 'does nothing. I hoped it would make Macs work
   Wend   ' "End this While Loop", because we've done all the frames.
   Range("C8").Select   ' "move the cursor to this cell on the notebook.
End Sub
```

Using the Command Button

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.

Now you can actually press the command button to see what it does.