**Intro**

So for my capstone I decided to take a look at this problem: if you had a model of a putting green and you knew where the hole was, how would you make a putt from a given position (i.e. how hard would you hit the ball and where would you aim)? And yes, this is actually about golf (aka the best sport in the world).

Because this involved ~~a lot of~~ too much math, I made Python do most of it. So to run the simulation, you’ll need Python, VPython, matplotlib (a graphing library), and my simulation file. Here are some downloads:

* Python (I only tested it for 2.7): <http://www.python.org/download/releases/2.7/>
* VPython: <http://www.vpython.org/>
* Matplotlib: <http://matplotlib.org/users/installing.html>
* Simulation file: look on the canvas page

Known problems:

* The initial speed predicted by the simulation is sometimes too high to be a reasonable putt.
* For the simulation, I use a bunch of really artificial-looking surfaces. I tried finding a model of a real green, but no luck.

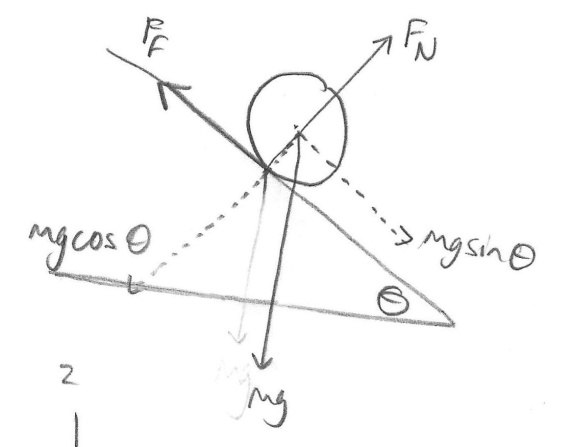
Please let me know if you find any other bugs.

**Description**

First off, I made some assumptions about the golf ball’s motion:

* On a real green, the surface is not totally smooth. The blades of grass are of different lengths, the grass grows in a certain direction, the green could be wet or dry, cold or hot, etc., all of which would affect the ball’s motion. For purposes of sanity, I just assumed that the surface *is* smooth and simplified the interactions between the ball and the grass as a “friction” force that acts opposite to the ball’s motion. I gave it a small coefficient of .04.
* The only other forces acting on the ball are gravity and the normal force.
* The ball is only rolling—there is no sliding or skidding.

Given all that, how do you model the ball’s motion? I started by thinking, ok, if you take a snapshot of the ball’s motion, it’s kind of like the inclined plane problem:



Except the difference is that the ball doesn’t have to be moving straight down the slope; it could be going into the page, out of the page, etc. But the gravity force vector still points straight down the slope, and the friction force vector is in the direction opposite of the ball’s velocity. So:

C:\Users\nalindquist\Downloads\CodeCogsEqn.png

And because the friction force is opposite the velocity,

C:\Users\nalindquist\Downloads\CodeCogsEqn (1).png

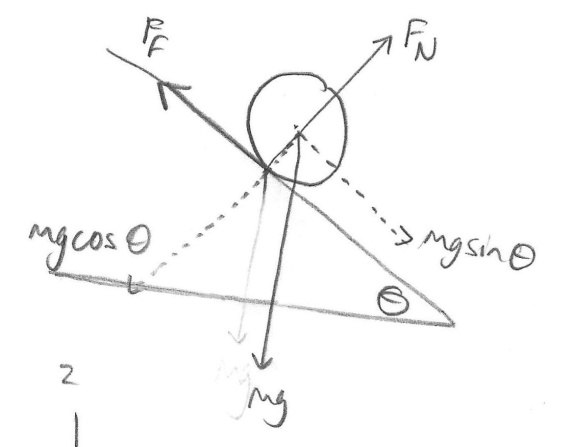
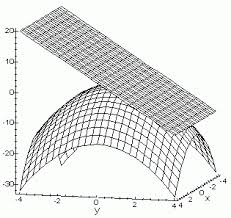
since C:\Users\nalindquist\Downloads\CodeCogsEqn.png is a unit vector in the direction of the velocity. (C:\Users\nalindquist\Downloads\CodeCogsEqn (3).png is the length of the velocity vector, so C:\Users\nalindquist\Downloads\CodeCogsEqn.pngis the original velocity vector scaled down to a length of 1).

The f­net equation becomes

C:\Users\nalindquist\Downloads\CodeCogsEqn (2).png,

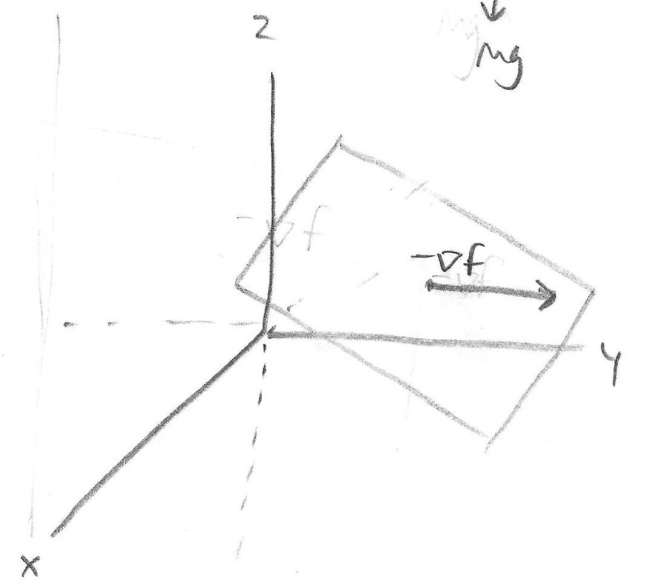
leaving only the gravity force (hahaha “only”… that’s a good one).

As for gravity… like the inclined plane problem, it has to act straight down the slope. But given that the surface is not a nice, smooth plane, what direction is “straight down the slope”? After listening to a certain awesome math teacher, I started thinking about the ball as if it was on the plane tangent to the surface at the ball’s current position:



Then say the surface is modeled by a function http://www.sciweavers.org/upload/Tex2Img_1387252481/eqn.png —i.e. for given x and y values, the surface of the green is at the corresponding z-value. From MVC (\*cry\*), the vector representing the direction of maximum decrease (aka the negative [gradient](http://betterexplained.com/articles/vector-calculus-understanding-the-gradient/), or the vector that points down the steepest part of the slope) is

http://www.sciweavers.org/upload/Tex2Img_1387250085/eqn.png, where http://www.sciweavers.org/upload/Tex2Img_1387253874/render.png is the [partial dervative](http://tutorial.math.lamar.edu/Classes/CalcIII/PartialDerivatives.aspx) with respect to x, and http://www.sciweavers.org/upload/Tex2Img_1387253954/render.png is the partial derivative with respect to y.

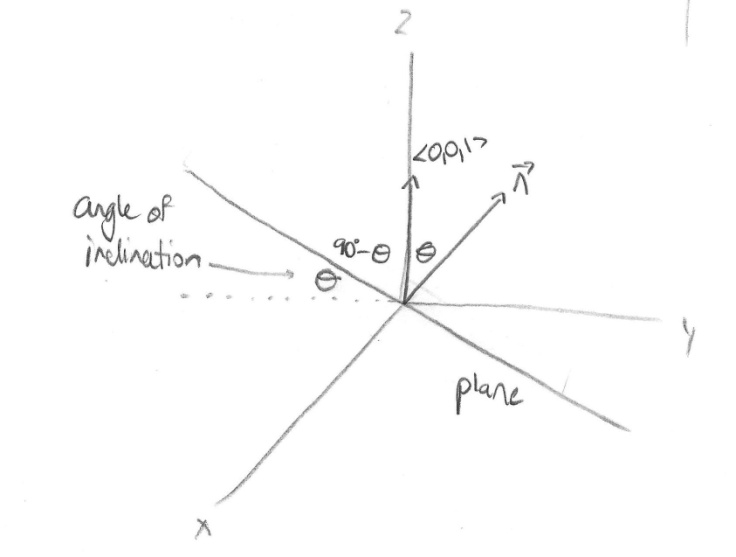


Since gravity acts in this direction, the gravity force is C:\Users\nalindquist\Downloads\CodeCogsEqn (6).png. (Only two dimensions here, x and y, because we know the z position of the ball at every point—it has to rest on the surface).

That means the final fnet equation is

C:\Users\nalindquist\Downloads\CodeCogsEqn (7).png.

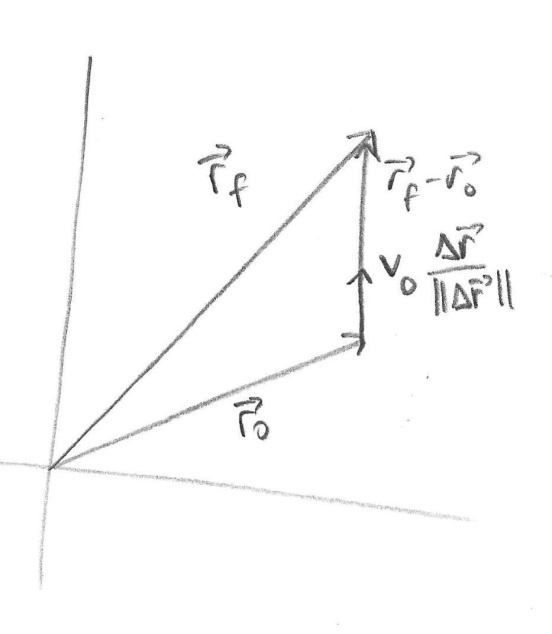
Whew… done. NOT. The angle’s still missing. Again thanks to a certain awesome math teacher, I learned that if you think about drawing the normal vector to the plane, then the angle of inclination of the plane is also the angle between the normal vector and the z-axis.



The cosine of the angle between any two vectors is http://www.sciweavers.org/upload/Tex2Img_1387250584/eqn.png.

Also, if http://www.sciweavers.org/upload/Tex2Img_1387250714/eqn.png, then the normal vector (again from MVC \*cry\*) is http://www.sciweavers.org/upload/Tex2Img_1387250869/eqn.png. So the angle between the normal and the z-axis, which is also the angle of inclination of the plane, becomes http://www.sciweavers.org/upload/Tex2Img_1387251116/eqn.png.

Now, here’s where Python comes in. Say the ball starts at position C:\Users\nalindquist\Downloads\CodeCogsEqn (10).png and the hole is at C:\Users\nalindquist\Downloads\CodeCogsEqn (11).png. That means the displacement vector from the start to the hole is C:\Users\nalindquist\Downloads\CodeCogsEqn (12).png.



Then Python does the rest. It makes a guess C:\Users\nalindquist\Downloads\CodeCogsEqn (13).png for the speed that the ball will have to start with to make it to the hole and tries the direction directly toward the hole. So the initial velocity of the ball in this case would be C:\Users\nalindquist\Downloads\CodeCogsEqn (14).png.

Python then calculates the net force on the ball at its start position using the fnet equation above. Given that acceleration is the net force divided by mass, Python can use this acceleration to calculate the change in velocity over a small time interval. (I defined that interval to be .001 s). Similarly, Python can use the velocity to find the change in position over that small time interval:

C:\Users\nalindquist\Downloads\CodeCogsEqn (16).png and

C:\Users\nalindquist\Downloads\CodeCogsEqn (18).png.

So, if the ball currently has some *i*th velocity, the (*i*+1)th velocity is the original velocity (the *i*th velocity) plus the acceleration times the change in time—which just approximates the ball’s motion as undergoing constant acceleration over a small time increment. A similar argument holds for the change in position, except the small time increment lets Python approximate the ball’s motion using the constant-velocity model.

Then the simulation runs… and the ball gets nowhere near the hole. Grrrr. In that case, Python detects whether the ball went past the hole and if it missed to the right of the whole. If it missed long, it decreases the initial speed estimate slightly, and if it missed to the right, it rotates the starting direction CCW (so instead of pointing at the hole, it would point to the left). Then the simulation runs over… and over… and over again until the ball passes within 5.4 cm (i.e. the radius) to the hole.

**References**

* Mrs. Moore. What more do you need? ☺
* Some information about a ball rolling on a surface: <http://www.billiards.colostate.edu/physics/Hierrezuelo_PhysEd_95_article.pdf>
* More info about rolling objects: <http://www.physics.ohio-state.edu/~gan/teaching/spring99/C12.pdf>