Golf Putt Project Summary

July 7, 2020

1 Premise

Given a surface $h(x, y)$, a ball on the surface at point (x_0, y_0), and a hole on the surface at point (x_h, y_h), what is the direction and speed that the ball must be hit so that it reaches the hole under the influence of gravity and friction. Furthermore, how should it be hit so that it has the minimum initial velocity needed to reach the hole.

2 User Guide

To use the app either run the code with the "images" folder in the same directory as the py file or download the "Golf App" folder and run "Basics.exe"

Once the app is running, a manual speed and angle can be set using the arrow keys, and a shot can be taken using the space bar. The number of guides which show where a given shot will go can be adjusted with the buttons under "Guides", and friction can be adjusted likewise. The Reset key will place the ball back to its initial spot and allow another shot to be taken. If a shot makes the hole, the details of the map and shot will printed to the console.

The three buttons under "Map Menu" allow you to switch between three preset maps, and the Create Map button will open a menu that allows you to build your own map. To a create a new map, first hit the clear button, and then add circle or line shaped hills using the corresponding buttons. After you are done painting the landscape, the update button will generate the surface you created as well as the corresponding height map and color bar. This may take a few a seconds, and too many features, especially lines, will increase the computational cost of the algorithms.

When visualize is set to "Shot", and an algorithm is run, the direction and magnitude of each shot will be shown as a line. This is best for seeing the Bisection and Smart Brute algorithms search the possible input space. When visualize is set to "Path", which can be toggled by clicking visualize, the actual path taken by the shots the algorithm looks at is shown. This is best for

seeing how the Gradient Descent algorithms converge toward the hole. Angle Res should be used to adjust how thoroughly the algorithms search the possible shot angles. Higher angle res will usually take longer to run but may result in a better solution.

3 Physics

A shot is simulated using numerical approximations of Newtonian equations of motion. In order to calculate both friction and weight, at each time step the surface is approximated by a tangent plane where $F_w = mg\sin(\theta)$ and $F_f = mg\mu\cos(\theta)$ can be applied for weight and friction respectively along one axis. Note that sliding friction is used rather than rolling friction so this is actually a hockey putt. Expressing these in terms of the slope of the plane rather than the angle we get that, in the x direction

$$F_w = mg\frac{\frac{\partial h}{\partial x}}{\sqrt{1+(\frac{\partial h}{\partial x})^2}} \textbf{ and } F_f = mg\frac{\mu}{\sqrt{1+(\frac{\partial h}{\partial x})^2}}.$$

For weight, the above equation is used for both the x and y direction independently, but since the magnitude of the velocity vector plays a role in how friction is applied, it becomes messy when done component wise. Instead, the magnitude of friction is calculated by

$$|F_f| = mg\frac{\mu}{\sqrt{1+(\nabla_u h)^2}}$$

where $\nabla_u h$ is the directional derivative of h in the direction of velocity. Using these to determine w and $|f|$, the acceleration vector caused by weight and the magnitude of friction, the position and velocity vectors are updated in the following way

$$x = x + v\Delta t + \frac{1}{2}w(\Delta t)^2 \textbf{ and } v = (v + w\Delta t)(1 - \frac{|f|}{|v|}\Delta t).$$

If $\frac{|f|}{|v|}\Delta t > 1$ velocity is set to 0.

4 Map Equations

The surfaces for the three built in maps are given by:

$$\begin{aligned} h_1(x,y) = 40\exp(-(\frac{(x-450)-0.001y^2}{150})^2) \\ +30\exp(-(\frac{x}{300})^2-(\frac{y}{300})^2) \\ +15\cos(\frac{x}{200})\sin(\frac{-y}{200})\exp(\frac{-(x+100)}{500}) \end{aligned} \quad (1)$$

$$h_2(x,y) = 15\exp(-(\frac{x}{400})^2 + (\frac{y}{200})^2)\cos((\frac{x}{100})^2 + (\frac{y}{100})^2) + 5\exp(-(\frac{x}{50})^2 + (\frac{y+400}{50})^2) + line((800,450),(400,950),40,100)(x,y) + circ(690,560,20,70)(x,y) \quad (2)$$

$$h_3(x,y) = 10\sin(\frac{x}{75})\sin(\frac{y}{75}) - \frac{y}{15} \quad (3)$$

The functions *circ* and *line*, which are used by the map editor and by the second map, are defined as follows:
For a circular hill centered at (a,b) with height h and width w,

$$circ(x,y) = h\exp(-(\frac{x-a}{w})^2 - (\frac{y-b}{w})^2)$$

and for a linear hill from (a_x,a_y) to (b_x,b_y) with height h and width w,

$$line(x,y) = \begin{cases} h\exp(\frac{(-(x-a_x)^2-(y-a_y)^2}{w^2/(m_1^2+1)}) & y < a_y + m_2(x-a_x) \\ h\exp(\frac{(-(x-b_x)^2-(y-b_y)^2}{w^2/(m_1^2+1)}) & y > b_y + m_2(x-b_x) \\ h\exp(-(\frac{(y-a_y)-m_1(x-a_x)}{w})^2) & \text{otherwise} \end{cases}$$

where $m_1 = (b_y - a_y)/(b_x - a_x)$ and $m_2 = -1/m_1$. Since both the circle and line are smooth functions, any map generated by summing these functions will be smooth.

5 Solutions

For many of the algorithms used to solve this problem, two important values are used. First is the Angle Res, which is how many equal sections each half of the angle input space ($[-90,0]$ and $[0,90]$) is separated into. The second is the minimum velocity. This is calculated based on the energy equation

$$\frac{1}{2}mv_i^2 + mgh_i - \int_C F_f \cdot dr = \frac{1}{2}mv_f^2 + mgh_f.$$

First assume that in the minimum speed case $v_f = 0$, and then bound the energy lost to friction by

$$\int_C F_f \cdot dr \geq \frac{mg\mu d(i,f)}{\sqrt{1 + (\frac{|h_1 - h_f|}{d(i,f)})^2}}$$

where $d(i,f)$ is the 2-dim euclidean distance from the initial to the final position. This is the expression for energy lost to friction assuming a straight path to the hole, and after using this instead of the line integral, a minimum velocity can be found by

$$v_i \geq \sqrt{2g(h_f - h_i + \frac{\mu d(i,f)}{\sqrt{1 + (\frac{|h_1 - h_f|}{d(i,f)})^2}})}$$

Note that in the code 1 is subtracted from this minimum. This is because the equations of motion are only approximated and energy may not be perfectly conserved, so a slightly lower bound may be needed in some cases.

5.1 Bisection and Smart Brute

Both of these algorithms work by searching through different hit speeds using the same algorithm to determine if a certain speed is possible. The determining algorithm is a bisection search that uses a simple heuristic to determine which direction it should search in. The heuristic works by first seeing if the closest point on the path to the hole was left or right of the hole. If left, it searches right, if right, it searches left. Then, it may flip the decision if the final x velocity has changed signs, and both the initial and final x velocities pass a certain threshold. Then, it will flip the decision once again if the final y velocity is negative. This heuristic can be fooled and so this does not guarantee that it will find a solution, however it works well in most situations and finds solutions fast. This determining algorithm is run for every section of the angle space determined by the Angle Res. In the standard Bisection solution, this determining algorithm will be run through input speeds using another bisection search, stopping once it is looking at velocities closer then 0.5. This of course assumes that if you can't find a solution at one speed, you can't find one in speeds below it. This is not true and Map 2 with Friction at 0.025 is a counter example. The Smart Brute, on the other hand, implements the minimum energy estimate and simply runs the determining algorithm on every integer velocity greater then the minimum, stopping once it finds the first one to have a solution.

5.2 Gradient Descent(GD)

Both of the GD algorithms use the fact that this can be expressed as an optimization problem and so one can search for a solution using GD. One part of the cost function that the GD algorithms minimize is the function $M(\theta, v)$, which returns the distance from the hole to the closest point on the path produced with the angle θ and speed v. For the standard GD, the cost function is

$$f(\theta, v) = \begin{cases} M(\theta, v) + \frac{v^2}{10} & \mu \leq 0.15 \\ M(\theta, v) & \text{otherwise} \end{cases}$$

and a descent step is determined by

$$X_i = X_{i-1} - 0.01(M(\theta, v) + 5)\nabla f*$$

where $\nabla f*$ is $\nabla f/|\nabla f|$ with the θ component scaled by $10/AngleRes$. Using this for the step allows for two things, step sizes proportional to the distance a shot is from one that hits the hole, as well as quickly changing θ. This was incorporated because the cost function can be very complicated at low frictions, so while normal gradient descent would radically change velocity when close a potential solution, this version mainly changes its angle. Because the steps

become chaotic near minimums, this algorithm starts at the minimum v and sweeps toward a correct θ, returning the first successful shot. Assuming the map doesn't have a local minimum the descent can get stuck on, this will find a solution very fast and be near to if not optimal. However, if it does get stuck it will start again with a higher v, eventually finding a solution. This solution may or may not be optimal depending on the map. One last note is that the cost function doesn't have the v^2 term in the high friction version. This is because at a high friction, the minimum estimate becomes very close to the minimum speed, so on average GD performs better just sweeping toward a solution with a slightly higher then minimum speed rather then trying to decrease both speed and distance.

5.3 Nadam GD

This algorithm uses an enhanced version of gradient descent that is more suited for the messy nature of this particular cost function. This algorithm combines both adaptive learning rates with Nesterov momentum. For this algorithm, the cost function used is

$$f(\theta, v) = (M(\theta, v))^2 + v^{3(1-\frac{\mu}{3})}$$

and descent is done with the following:

$$\begin{aligned}
m_i &= \gamma_1 m_{i-1} + (1-\gamma_1)\nabla f \\
n_i &= \gamma_2 n_{i-1} + (1-\gamma_2)(\nabla f)^2 \\
M_i &= \gamma_1 m_i/(1-\gamma_1^{i+1}) + ((1-\gamma_1)\nabla f/(1-\gamma_1^i)) \\
N_i &= \gamma_2 n_i/(1-\gamma_2^i) \\
X_i &= X_{i-1} - \frac{3}{\sqrt{N_i + 0.00001}} M_i
\end{aligned}$$

where $\gamma_1 = 0.75$ and $\gamma_2 = 0.9$ and $(\nabla f)^2$ is the component wise square of the gradient. Using a form of GD that incorporates the adaptive step size along with the Nestorov momentum means that unlike the other GD solution, this one can continue to search for solutions when it has already found a minimum. Because of this, Nadam GD won't return a solution until it has gone through all its iterations from a starting point that gave solutions. This allows the algorithm to find the region in the input space that gives a valid solution, and slowly descent toward the minimum minimum velocity without getting thrown off by the bumps in the cost function. On simple surfaces, this algorithm is the most consistent in finding the optimal solution quickly, however this is also the most susceptible to getting stuck in a local minimum that is far from the best solution. For more technical information on Nadam GD read https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ.