Alen Sugimoto
Mr. Lee, Teacher
IB Math HL 12
March 13, 2020

Base Running Optimization

In this math exploration, I am going to be finding an approximation of the optimal base running path from home to second base specific to my physical abilities on a baseball field. The task for optimization is focused only from home to second base, because base runners advance two bases at a time more often than three or four and the optimal path from a base to an adjacent one is just a straight line. This topic really excited me because I played baseball for more than eight years and wanted to find out how the optimal path approximately compares with the path many of my coaches had taught me to use (see Figure 1). The thought that this particular path, which initially makes a straight line parallel to the baseline (a line connecting two adjacent bases), is ideal is a common misconception; this makes sense because we can decrease the time it takes to run along the path by simply replacing the curve in the x-interval $[0, 75]$ with a straight line, which makes the path shorter in distance and contain less turns. This exploration is significant because all of my former teammates in the baseball organization I played for are using paths similar to the one in Figure 1 when trying to reach second base, which means that, by finding an approximation of the optimal path, I can help them become much more efficient base runners. Additionally, although there is an article[1] that optimized a base runner's path already, that article applies a level of mathematics that is not appropriate for this IA, and thus, what makes this exploration unique is that it uses a much simpler method that can be understood by my peers. My method of finding an approximation of the optimal path is summarized as follows:

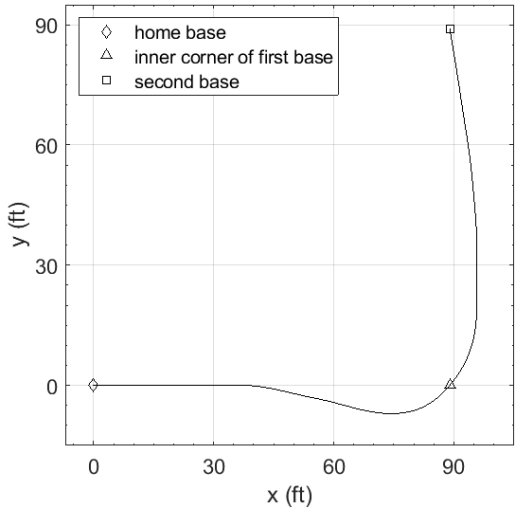


Fig. 1. The shape of the path I had been taught to use when sprinting from home to second base.

1. Collect data that lets me determine the approximate time I would take to run along a given path to second base;
2. Make a function in MATLAB[2] that does the following:
   - 2.1. Reasonably condense the number of possible paths we can evaluate as much as possible;
   - 2.2. Produce a path that is a possible good approximation of the optimal path;
   - 2.3. Plot a speed versus distance graph of me running on that path;
   - 2.4. Find the time I would spend running along the path using the graph in step 2.3;
   - 2.5. Continuously repeat steps 2.2 to 2.4 until obtaining a statistically significant large sample of paths; and
   - 2.6. Display the path with the smallest calculated time in the sample. This path will be the approximation of the optimal path.

1. Initial Analysis

Let's express time t, the variable to be minimized, in terms of distance s and instantaneous speed v.[3]

By definition, $$v = \frac{ds}{dt}$$

$$\therefore \frac{dt}{ds} = \frac{1}{v}$$

$$\Rightarrow t = \int \frac{1}{v} ds \qquad (1)$$

Equation (1) prompts me to find relationships between my speed and distance for many different paths so that the time it takes me to run along them can be calculated and compared to find the best path.

2. Data Collection

On a ninety-foot baseball diamond, I collected sets of continuous data in order to find my maximum speed while turning and running straight and my maximum and minimum acceleration. I timed myself with a video analysis app called *Coach's Eye* and used my footprints to find the distance I had travelled accurately.

For the first set of data, I sprinted in a straight line, and once I felt as if I reached my maximum running speed, I decelerated to a stop as fast as I could. The data points in the graph in Figure 2 are collected with every stride, and their coordinates are shown in the Appendix. My maximum speed running in a straight line was obtained by finding the slope of a linear trend line for the data points in the time interval $[3.11, 8.22]$ (see Figure 3). This interval was selected because all the footprints after $8.22\,\text{s}$ showed that I pushed my heels to the ground, indicating decelerating, and my stride lengths made at and after $3.11\,\text{s}$, which fluctuated between $65\,\text{in}$ and $72\,\text{in}$, distinguished themselves from the prior strides, which were constantly increasing in length.

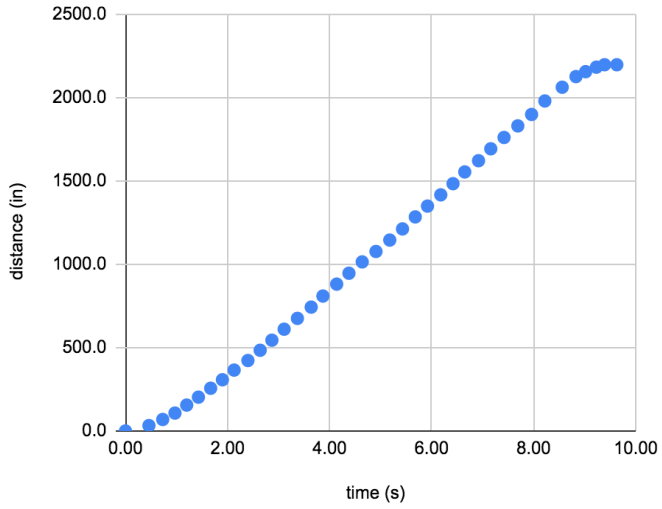


Fig. 2. A distance vs time graph of a straight sprint with a standing start and a rushed stop.

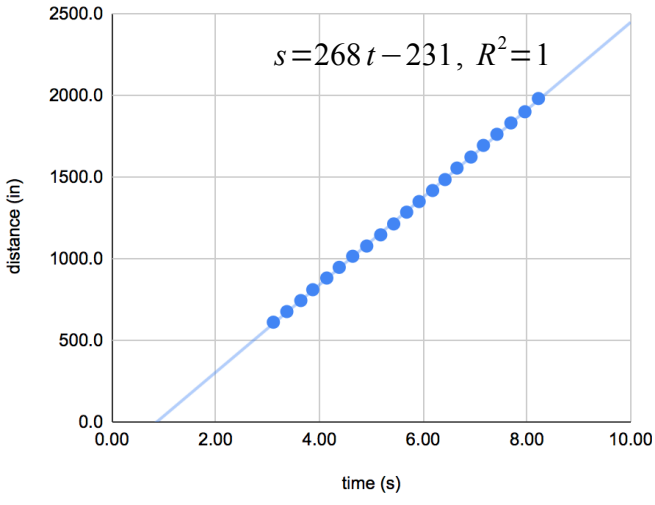


Fig. 3. A distance vs time graph of a straight sprint in the time interval where speed is constant.

Note that throughout this exploration, feet and seconds are used as measurement units. Therefore, my maximum speed was stored as $22.\overline{3}$ ft/s instead of 268 in/s.

My average acceleration and deceleration were calculated on the time intervals $[0, 3.11]$ and $[8.22, 9.63]$. With the assumption that the magnitude of jerk is always zero whenever I sprint, they are made equivalent to my maximum and minimum accelerations, a_{max} and a_{min}.

Therefore, using the formula $\bar{a} = \frac{\Delta v}{\Delta t}$,

$$a_{max} = \left(\frac{67.0}{3}\,\text{ft s}^{-1}\right)\frac{1}{(3.11\,\text{s})} = 7.18\,\text{ft s}^{-2}\ \text{(3SF)}$$

$$a_{min} = \left(-\frac{67.0}{3}\,\text{ft s}^{-1}\right)\frac{1}{(9.63\,\text{s} - 8.22\,\text{s})} = -15.8\,\text{ft s}^{-2}\ \text{(3SF)}$$

The Appendix shows that distance is the same at both 9.39 s and 9.63 s; however, it does not mean I stopped moving at 9.39 s because the distances are calculated using my footprints and my centre of mass was still moving at the time.

For the second set of data, I sprinted in circles of varying radii at constant speeds. The distance of every timed lap was the sum of the circumference of a circle of radius r and the distance $|\boldsymbol{\Delta s}|$ between the starting line and the last footprint that passed it. Using the radii r, lap times Δt, and magnitudes of displacement $|\boldsymbol{\Delta s}|$ collected (see Table 1), my average speed $\bar{v}$ for each circle was calculated as follows to produce the graph in Figure 4:

$$\bar{v} = \frac{2\pi r + |\boldsymbol{\Delta s}|}{\Delta t}$$

Table 1

The continuous dataset collected to find the relationship between my running speed and the radius of a circle I run along

Radius (ft)	Time (s)	Displacement (in)	Speed (ft/s)
1.0	1.58	3.0	4.1
2.0	1.98	11.0	6.8
3.0	2.05	5.5	9.4
6.0	3.29	11.5	12
9.0	4.36	24.0	13
12.0	5.29	43.0	14.9
15.0	6.03	3.0	15.7
18.0	6.67	9.5	17.1
21.0	7.11	21.5	18.8
24.0	7.66	26.5	20.0

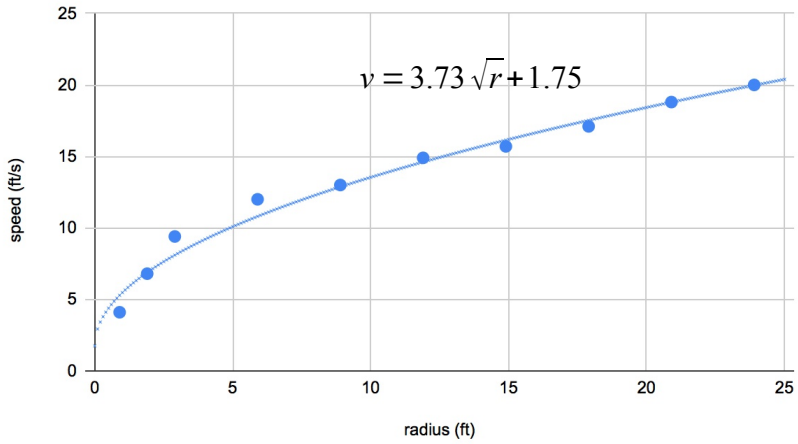


Fig. 4. A speed vs radius graph of circular sprints.

The equation of the trend line for all the data points, as written in (2), was used to relate instantaneous speed v and radius r, assuming that $v \propto \sqrt{r}$ from the centripetal acceleration formula $a_c = \frac{v^2}{r}$.

$$v = 3.73\sqrt{r} + 1.75 \tag{2}$$

However, (2) does not make sense for large values of r because as $r \to \infty$, it should be that $v \to 22.\overline{3}$. A piecewise function f that takes this into account is found by doing the following:

$$v = \frac{268}{12} = 22.3 \text{ (3SF)}$$

$$3.73\sqrt{r} + 1.75 = \frac{67.0}{3}$$

$$\Rightarrow r = 30.5 \text{ (3SF)}$$

$$\therefore f(r) = \begin{cases} 3.73\sqrt{r} + 1.75, & 0 \le r < 30.5 \\ 22.3, & r \ge 30.5 \end{cases} \tag{3}$$

3. The MATLAB Function

I wrote a function in MATLAB that produces possibly-optimum base running paths and compares them to find the best one—the one that takes the least time for me to run along. From now on, this function will always be referred to as a MATLAB function to avoid any confusion.

3.1. Setting Restrictions

Because it is computationally expensive to search for the best approximation of the optimal path in an infinite number of possible paths from home to second base, we are going to condense the population in a way so that a reasonably good approximation can be found in a short time.

My first step in doing that is ignoring paths that are certainly bad approximations of the optimum. As indicated before, the path I had been taught to use is an example of a bad approximation because it makes a right turn. Therefore, paths with right turns will all be ignored.

My second step is to plot nodes from home to second base that paths, represented by cubic polynomials, must pass through. The start and end nodes will be located at home and second base respectively. The node at first base will be placed on the corner of first base closest to the pitching mound for the reason that runners can push off of it better than at any other place on the base. The positioning of the rest of the nodes is shown in Figure 5.

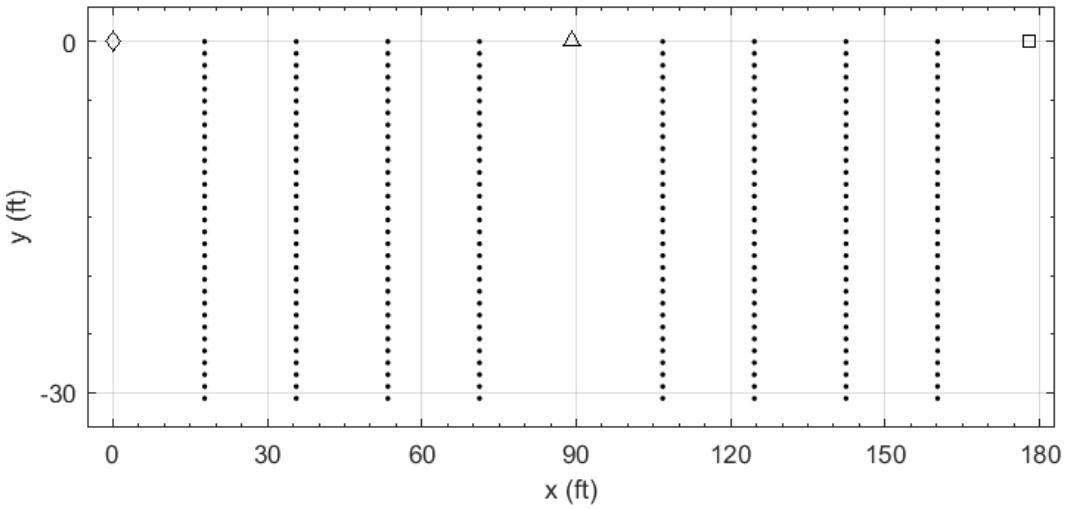


Fig. 5. The placement of nodes for paths to pass through. In the MATLAB function, the lone nodes at home, first, and second are lined up from left to right, respectively, to ensure that paths can be defined by functions.

Notice how the bases are lined up horizontally. All the figures from now on will have nodes positioned as one would expect on a baseball diamond with second base directly above first. However, keep in mind that the MATLAB function constantly has the nodes positioned in the way shown in Figure 5 to ensure that paths can be defined by functions.

Between the lone/base nodes, which are 89 ft apart due to positioning a node at the corner of first base, are 4 columns of 31 evenly spaced nodes. I chose those dimensions because they create the desired distances between the evenly spaced nodes: exactly 17.8 ft horizontally and around 1 ft vertically. The distances are selected so that the number of different paths is reduced as much as possible while still allowing polynomial curves to approximate the optimum path well, knowing that it is smooth between nodes and easy to run along. I also feel that I can run along a path very close to the one I want even if I need to pass through one node per every column of nodes positioned as in Figure 5, confirming that my decision is reasonable.

I made it so that the nodes span a width of 30.5 ft vertically, which is the value of the smallest radius such that $f(r) = 22.3\ \text{ft s}^{-1}$, because the widest reasonable turn I would have to make is when I sprint in a straight line to first base and run through it at maximum speed without slowing down until I reach second base. To minimize distance in this case, the path would have to consist of a quarter-circle with a radius of 30.5 ft immediately after first base, and thus, the width is restricted to that number.

3.2. Producing a Path

Every iteration of the MATLAB function produces one path, calculates the time it would take for me to run along it, and compares the result with the best time found so far.

Let's say we are in one iteration of the MATLAB function. The iteration starts by selecting one node per column of nodes randomly (see Figure 6), beginning from the leftmost column and considering the restrictions made in the previous section. Let n_i with coordinates (x_i, y_i) be the $(i+1)$th node counting from the left such that n_0 is the first node, positioned at home base. Remember that, in the MATLAB function, the base nodes all lie on the x-axis, that is, $y_0 = y_5 = y_{10} = 0$, and all the nodes are positioned such that $x_i = 17.8i$.

To satisfy the restrictions made, the nodes are selected so that the straight lines connecting them avoid making right turns. For each column starting from the leftmost one, the MATLAB function selects a random node n_{i+1} such that the line through it and n_i has a slope that is greater than or equal to the slope of a line through n_i and n_{i-1} and that is less than or equal to another through n_i and the approaching base node. The nodes n_1 and n_6 are exceptions; for n_1, any node in the first column can be randomly chosen to represent it since the node n_{-1} does not exist, and for n_6, the line through it and n_5 should just make a slope greater than or equal to the negative reciprocal of the slope of the line through n_5 and n_4.

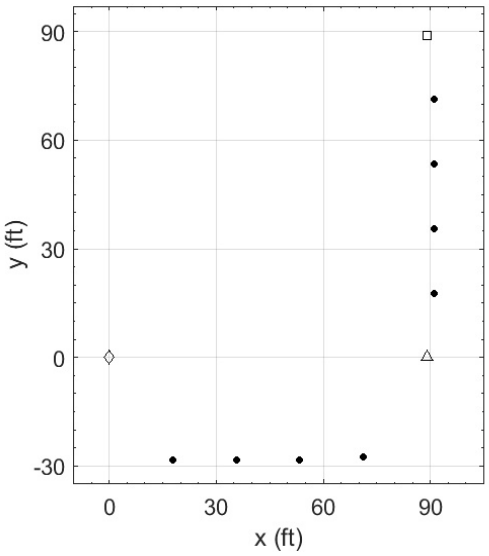


Fig. 6. Randomly selected nodes considering all previously made restrictions.

Before drawing a smooth curve through all the selected nodes, the MATLAB function makes another set of nodes with different y-coordinates so that they are not so awkward to run through. For a node n_i, its new y-coordinate is equal to $\frac{y_{i-1}+y_i+y_{i+1}}{3}$.[4] By finding the new y-coordinates for all the nodes except n_0, n_5, and n_{10} (the base nodes should not move since bases are stationary), we get a smoother-looking series of nodes (see Figure 7). This is done because I am assuming that the optimal path is not composed of tight turns, and curves passing through raw nodes will have tight turns (see Figure 8). Therefore, if we continue on with the nodes in Figure 6, the MATLAB function may end up finding a very different approximation due to compensating for the problem.

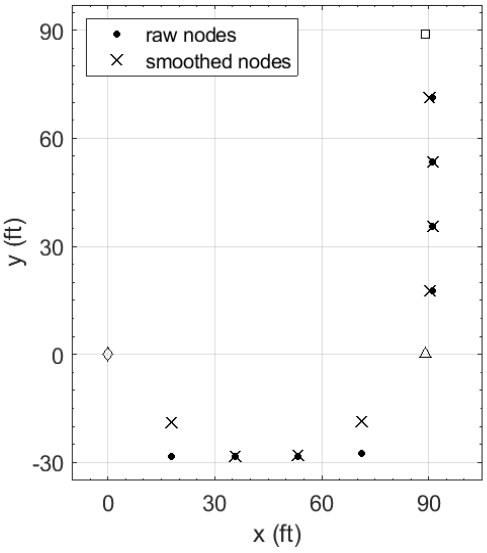


Fig. 7. A comparison between the smoothed nodes and the raw nodes.

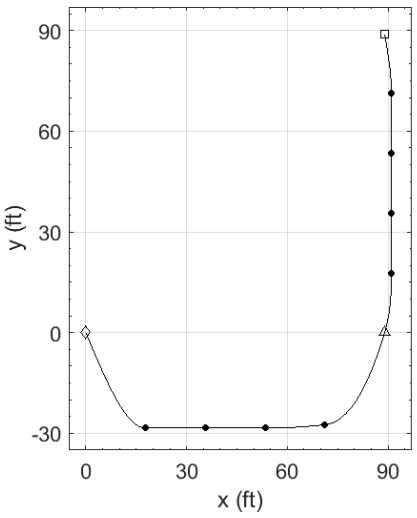


Fig. 8. A curve through the raw nodes.

Since n_5 has not been moved and will be in the middle of a path, we should try to smoothen it as well. But what if I do not (see Figure 9)? The produced path will inevitably have right turns. Therefore, the MATLAB function will go about smoothing n_5.

Knowing that three nodes must be equally spaced out horizontally when finding the average of their y-coordinates, we will rotate n_4, n_5, and n_6 concurrently in some way. Note that the MATLAB function is still using the raw nodes, which are in Figure 6.

Let's represent n_4, n_5, and n_6 with the complex numbers z_4, z_5, and z_6 such that z_5 is at the origin of an Argand diagram:

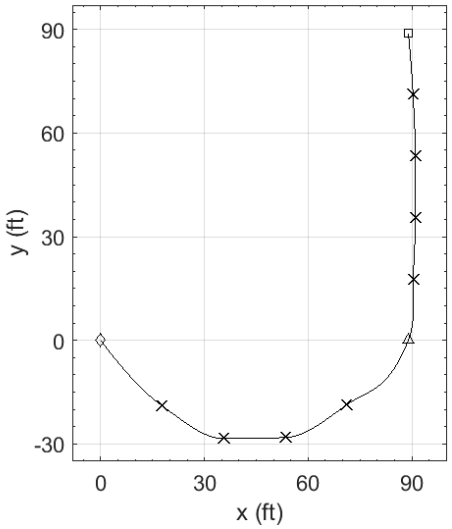


Fig. 9. A curve through the current smoothed nodes.

$$z_4 = x_4 - 89 + iy_4 = (71.2) - 89 + i(-27.45) = -17.8 - i\,27.45$$
$$z_5 = x_5 - 89 + iy_5 = (89) - 89 + i(0) = 0$$
$$z_6 = (x_6 - 89 + iy_6)\operatorname{cis}\frac{\pi}{2} = -y_6 + i(x_6 - 89) = -(-2.0\overline{3}) + i((106.8) - 89) = 2.03 + i\,17.8 \text{ (3SF)}$$

Notice that n_6 is turned 90º counterclockwise about n_5 to get z_6. I do this to correct for the fact that the MATLAB function stores the location of the nodes such that the base nodes make a straight line when in fact there is a 90º turn at first base. Now, I will find a general equation for the MATLAB function to find how many θ radians are required to turn both z_4 and z_6 about the origin such that the origin is centred between them relative to the real axis.

$$\begin{aligned}
& -\operatorname{Re}(z_4 \operatorname{cis}\theta) = \operatorname{Re}(z_6 \operatorname{cis}\theta) \\
\Rightarrow\ & -\operatorname{Re}[(-17.8 + iy_4)(\cos\theta + i\sin\theta)] = \operatorname{Re}[(-y_6 + i\,17.8)(\cos\theta + i\sin\theta)] \\
\Rightarrow\ & -(-17.8\cos\theta - y_4\sin\theta) = -y_6\cos\theta - 17.8\sin\theta \\
\Rightarrow\ & (y_4 + 17.8)\sin\theta = (-y_6 - 17.8)\cos\theta \\
\Rightarrow\ & \tan\theta = -\frac{y_6 + 17.8}{y_4 + 17.8} \\
\Rightarrow\ & \theta = -\arctan\left(\frac{y_6 + 17.8}{y_4 + 17.8}\right) \leftarrow \text{the general equation} \\
& = -\arctan\left(\frac{(-2.0\overline{3}) + 17.8}{(-27.45) + 17.8}\right) \\
& = 1.02 \text{ (3SF)}
\end{aligned}$$

Therefore, the new complex numbers w_4, w_5, and w_6 that allow the MATLAB function to use the average formula are

$$w_4 = z_4 \operatorname{cis}\theta = (-17.8 - i\,27.45)\operatorname{cis}(1.0215...) = 14.1 - i\,29.5 \text{ (3SF)},$$
$$w_5 = z_5 \operatorname{cis}\theta = (0)\operatorname{cis}\theta = 0 \text{ since } z_5 \text{ is at the origin, and}$$
$$w_6 = z_6 \operatorname{cis}\theta = (2.0\overline{3} + i\,17.8)\operatorname{cis}(1.0215...) = -14.1 + i\,11.0 \text{ (3SF)}.$$

Now, similar to what we have done with the other nodes, the smoothed complex number k_5 for n_5 is found by doing the following:

$$\mathrm{Im}(k_5)=\frac{\mathrm{Im}(w_4+w_5+w_6)}{3}=\frac{(-29.51...)+(0)+(11.02...)}{3}=-6.161...$$
$$\therefore k_5=-i\,6.16 \text{ (3SF)}$$

With this, the MATLAB function can find the new coordinates of n_5 that will fit in nicely with the other nodes by rotating k_5 back $-\theta$ radians and finding the real and imaginary parts of the resulting complex number.

Let $\begin{pmatrix}\Delta x_5\\ \Delta y_5\end{pmatrix}$ be the translation vector that will translate n_5 to its new location.

$$\Delta x_5=\mathrm{Re}[k_5\,\mathrm{cis}(-\theta)]=-(-6.161...)\sin(-1.0215...)=-5.2\bar{5}=-5.26 \text{ (3SF)}$$
$$\Delta y_5=\mathrm{Im}[k_5\,\mathrm{cis}(-\theta)]=(-6.161...)\cos(-1.0215...)=-3.21\bar{6}=-3.22 \text{ (3SF)}$$

However, if I move n_5, it will not be at the corner of first base anymore. Therefore, instead of moving n_5, the MATLAB function will move all the other nodes such that the resulting trend is similar in shape to the one that would have been made after moving n_5. The translation vector the MATLAB function uses is $(1-\frac{|5-i|}{5})(5.26\,\hat{\mathbf{i}}+3.22\,\hat{\mathbf{j}})$, where i is the subscript of a node n_i that is to be translated, and thus, $i\neq 5$. The translation vector will ensure that the nodes n_0 and n_{10} do not move and a smooth trend of nodes will be produced. After the translations, the final series of nodes are found, which will always be used from now until the end of this iteration, shown in Figure 10.

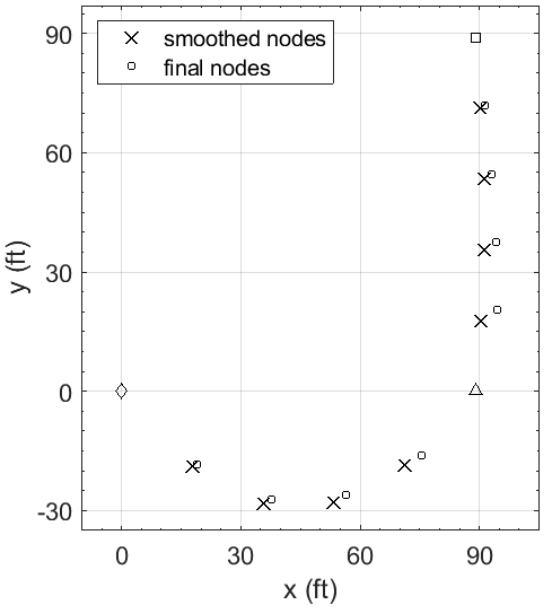


Fig. 10. The final series of nodes after translating the smoothed nodes.

The node manipulations are done in an effort to make all the paths produce possible good approximations of the optimum, or in other words, paths that are always turning left or going straight. Although manipulating the nodes in this particular way might not give us the best result, it will certainly help the MATLAB function find better paths faster and give us an approximated optimum one that baseball players and I can learn from.

After the smoothing, the MATLAB function uses PCHIP (Piecewise Cubic Hermite Interpolating Polynomial) functions, which are continuous and differentiable for all $x\in\mathbb{R}$ in their domains and each define multiple polynomial curves that can pass through all the newly produced nodes. One PCHIP function is used between home and first and another between first and second. Every curve between two adjacent nodes is represented by a single sub-function. The sub-functions are cubic polynomials of the form $y=a(x-x_i)^3+b(x-x_i)^2+c(x-x_i)+d$, where a, b, c, and d are coefficients and x_i is the x-coordinate of the $(i+1)$th node a sub-function begins at.

The PCHIP curve through the nodes in Figure 11 is smooth, similar to the path someone would take given the nodes that they must run through, which is why I decided to use the function. Since we are using two separate PCHIP functions, it is expected that the point at n_5 is not differentiable. Therefore, I must correct the sub-functions such that $\lim_{x\to 89^-} P'(x)=\lim_{x\to 89^+} -P'(x)^{-1}$, where P is a piecewise function of the PCHIP functions.

To make P differentiable when $x=89$, the MATLAB function will find new equations to define the sub-functions applying to the intervals $\left[x_4, x_5\right[$ and $\left[x_5, x_6\right[$. To closely follow the current shape of the path at n_5, I decided to have the sub-functions meet with a tangent parallel to the direction vector $\boldsymbol{d}$ such that $\boldsymbol{d}=\hat{\boldsymbol{d}}_1+\hat{\boldsymbol{d}}_2$, where $\hat{\boldsymbol{d}}_1$ and $\hat{\boldsymbol{d}}_2$ are unit vectors parallel to the tangents with slopes $\lim_{x\to 89^-} P'(x)$ and $\lim_{x\to 89^+} -P'(x)^{-1}$ respectively.

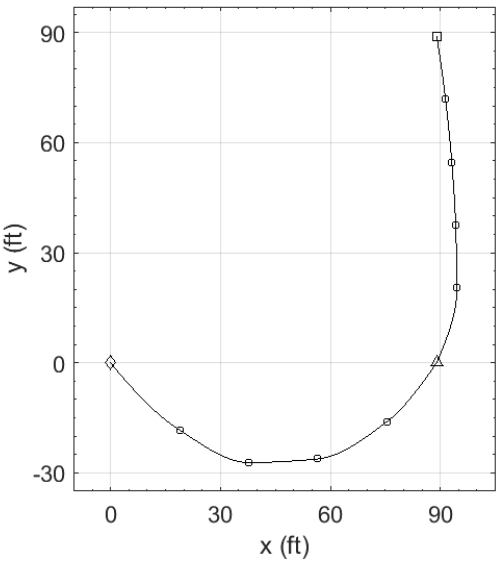


Fig. 11. Two PCHIP curves defining a base runner's path from home to second base.

In this iteration,

$$\text{if } x_i\le x<x_{i+1},\ P(x)=a(x-x_i)^3+b(x-x_i)^2+c(x-x_i)+d$$
$$\Rightarrow P'(x)=3a(x-x_i)^2+2b(x-x_i)+c$$

$$\therefore \lim_{x\to 89^-} P'(x)=\lim_{x\to 89^-}\left[3(-8.52...\times 10^{-4})(x-x_4)^2+2(0.430...)(x-x_4)+(0.754...)\right]$$
$$=3(-8.52...\times 10^{-4})(89-75.4...)^2+2(0.430...)(89-75.4...)+(0.754...)$$
$$=1.45 \text{ (3SF)}$$

$$\lim_{x\to 89^+} P'(x)=\lim_{x\to 89^+}\left[3(2.72...\times 10^{-4})(x-x_5)^2+2(0.00230...)(x-x_5)+(-0.4328...)\right]$$
$$=3(2.72...\times 10^{-4})(89-89)^2+2(0.00230...)(89-89)+(-0.4328...)$$
$$=-0.433 \text{ (3SF)}$$

$$\therefore \boldsymbol{d}_1=\begin{pmatrix}1\\1.451...\end{pmatrix} \tag{4}$$
$$|\boldsymbol{d}_1|=\sqrt{1+1.451...^2} \tag{5}$$
$$\boldsymbol{d}_2=\begin{pmatrix}0.4328...\\1\end{pmatrix} \tag{6}$$
$$|\boldsymbol{d}_2|=\sqrt{1+0.4328...^2} \tag{7}$$

Using equations (4) to (7) and that $\hat{\boldsymbol{d}}_1=\frac{1}{|\boldsymbol{d}_1|}\boldsymbol{d}_1$ and $\hat{\boldsymbol{d}}_2=\frac{1}{|\boldsymbol{d}_2|}\boldsymbol{d}_2$,

$$\boldsymbol{d}=\frac{1}{\sqrt{1+1.451...^2}}\begin{pmatrix}1\\1.451...\end{pmatrix}+\frac{1}{\sqrt{1+0.4328...^2}}\begin{pmatrix}0.4328...\\1\end{pmatrix}=\begin{pmatrix}0.96\\1.74\end{pmatrix} \text{ (2DP)}$$

With $\boldsymbol{d}$ calculated, I will find new functions to replace the sub-functions surrounding n_5 so that they will merge smoothly with tangents parallel to $\boldsymbol{d}$.

Let g_1 be a function that is to replace the sub-function applying to the interval $\left[x_4, x_5\right[$.

$$g_1(x)=a_1(x-x_4)^3+b_1(x-x_4)^2+c_1(x-x_4)+d_1$$

$$\begin{cases} g_1(x_4)=y_4=-16.06... \\ \lim\limits_{x\to x_4^+} g_1'(x)=\lim\limits_{x\to x_4^+} P'(x)=0.7541... \\ g_1(x_5)=y_5=0 \\ \lim\limits_{x\to x_5^-} g_1'(x)=\dfrac{1.741...}{0.964...} \end{cases} \Rightarrow \begin{cases} d_1=-16.06... \\ c_1=0.7541... \\ a_1(13.59...)^3+b_1(13.59...)^2+c_1(13.59...)+d_1=0 \\ 3a_1(13.59...)^2+2b_1(13.59...)+c_1+0=\dfrac{1.741...}{0.964...} \end{cases}$$

Solving the simultaneous equation, we get

$$a_1=0.00106,\ b_1=0.0170,\ c_1=0.754,\ d_1=-16.1 \text{ (3SF).}$$

Let g_2 be a function that is to replace the sub-function applying to the interval $\left[x_5, x_6\right[$.

$$g_2(x)=a_2(x-x_5)^3+b_2(x-x_5)^2+c_2(x-x_5)+d_2$$

$$\begin{cases} g_2(x_5)=y_5=0 \\ \lim\limits_{x\to x_5^+} g_2'(x)=-\left(\dfrac{1.741...}{0.964...}\right)^{-1} \\ g_2(x_6)=y_6=-5.56 \\ \lim\limits_{x\to x_6^-} g_2'(x)=\lim\limits_{x\to x_6^-} P'(x)=0 \end{cases} \Rightarrow \begin{cases} d_2=0 \\ c_2=-\dfrac{0.964...}{1.741...} \\ a_2(20.37\overline{3})^3+b_2(20.37\overline{3})^2+c_2(20.37\overline{3})+d_2=-5.56 \\ 3a_2(20.37\overline{3})^2+2b_2(20.37\overline{3})+c_2+0=0 \end{cases}$$

The coefficients were calculated to be

$$a_2=-1.96\times10^{-5},\ b_2=0.0142,\ c_2=-0.554,\ d_2=0 \text{ (3SF).}$$

Replacing the original sub-functions with g_1 and g_2 allows a smooth curve to be made that represents a path (see Figure 12). The process of smoothing the curve at n_5 is automated in the MATLAB function by, in each iteration, using stored values for the coordinates of final nodes and the coefficients of sub-functions, along with a linear system of equations solver in MATLAB.

Since I have decided on a specific way of setting the value of $P'(x)$ at $x=89$, the curve may not be the best it can be for the selected nodes. However, due to the fact that we smoothed n_5, there should not be a large difference between what is picked and the best way to get through the base. And that can be proven by looking at Figure 12; with the graph, we can see that there is only a small range of derivatives that could yield an optimal path through n_5.

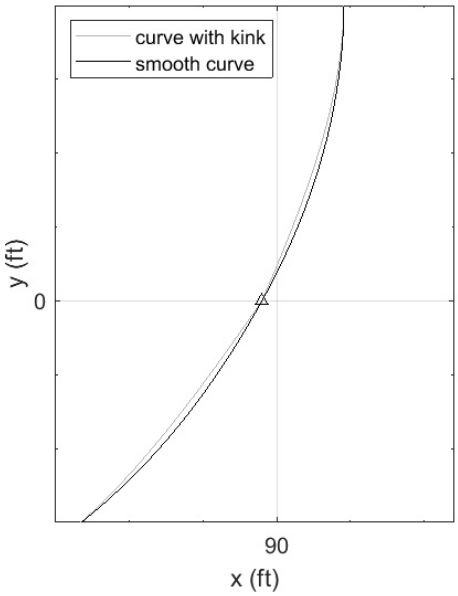


Fig. 12. A graph of a curve at first base before and after replacing sub-functions.

3.3. Plotting a Speed Versus Distance Graph

As we have seen, my running speed is dependent on the radius of a circle I run along, which means that speed is dependent on the curvature of a path. And since curvature can vary at different points on a path defined by PCHIP functions, curvature is dependent on their argument x. Therefore, we need to find x in terms of distance s so that an equation relating v and s can be found.

To find a relationship between s and x, we need to derive the arc length formula:[5]

The arc length Δs between two points on a curve can be first approximated by finding the length of a line segment connecting those points.

$$\Delta s \approx \sqrt{\Delta x^2 + \Delta y^2},\ \Delta x > 0$$
$$\Rightarrow \Delta s \approx \Delta x \sqrt{1 + \left(\frac{\Delta y}{\Delta x}\right)^2}$$

If the two points are infinitely close to each other, the arc length can be found exactly.

$$\lim_{\Delta x \to 0} \Delta s = \lim_{\Delta x \to 0} \Delta x \sqrt{1 + \left(\frac{\Delta y}{\Delta x}\right)^2}$$
$$\Rightarrow ds = dx \sqrt{1 + \left(\frac{dy}{dx}\right)^2}$$

Integrating both sides, we get $s = \int \sqrt{1 + \left(\frac{dy}{dx}\right)^2}\, dx$.

Therefore, the distance I travel on a path defined by P in terms of x is

$$s = \int_0^x \sqrt{1 + P'(x)^2}\, dx \tag{8}$$

Since P is a piecewise cubic polynomial function, there might be a quartic polynomial under the radical sign, and thus, numerical integration will be used and a general equation for x in terms of s cannot be found. As an alternative to finding x in terms of s analytically, let $s = \{s_0, s_1, s_2, \ldots, s_{N-1}\}$ and $x = \{u_0, u_1, u_2, \ldots, u_{N-1}\}$, where N is the number of elements in each set. The set s has evenly spaced elements in the range from zero to the total arc length of a path inclusive such that the difference between adjacent elements is approximately 0.06 ft (the reasonableness of the step length will be looked at later). It is approximate because N must be a whole number and s_0 and s_{N-1} must equal to zero and the total arc length respectively; however, $s_1 = 0.06$ ft and $s_{N-2} = s_{N-1} - 0.06$ ft (I will explain why they are exactly 0.06 ft from s_0 and s_{N-1} later).

To prepare the MATLAB function to solve for u_j, the $(j+1)$th element of x, given s_j, the $(j+1)$th element of s, it is going to first find the distance of every node from n_0. We do this so that the MATLAB function will only need to worry about a single sub-function of the PCHIP functions when solving for u_j. Let L_i be the length of the path defined by P between n_i and n_0.

$$L_0 = 0$$
$$L_i = \sum_{k=1}^{i} \int_{x_{k-1}}^{x_k} \sqrt{1 + P'(x)^2}\, dx,\ i \neq 0$$

After the results are stored, equation (9) below is used along with a root-finding algorithm to find the values of all the elements of x.

$$s_j = \int_0^{u_j} \sqrt{1 + P'(x)^2}\, dx, \text{ from equation (8)}$$
$$\Leftrightarrow 0 = \int_0^{u_j} \sqrt{1 + P'(x)^2}\, dx - s_j$$
$$\Rightarrow 0 = L_i + \int_{x_i}^{u_j} \sqrt{1 + P'(x)^2}\, dx - s_j, \text{ where } s_j \in \left[L_i, L_{i+1} \right[\quad (9)$$

After the MATLAB function finds the values of all the elements of x, the curvatures of P will also be calculated. The curvature formula can be derived as follows:[6]

Curvature, denoted κ, is defined as the reciprocal of radius: $\kappa = \frac{1}{r} = \frac{\theta}{s}$. For a curve in general, κ can be approximated by using a circle that touches the curve at two points.

$$\kappa \approx \frac{\Delta\theta}{\Delta s},$$

where $\Delta\theta$ is the angle that the circle makes between the points, and Δs is the arc length of the curve between the points. Curvature κ at a point is found exactly by using a limit:

$$\kappa = \lim_{\Delta s \to 0} \frac{\Delta\theta}{\Delta s} = \frac{d\theta}{ds} = \frac{d\theta}{dx}\frac{dx}{ds}$$
$$= \frac{d\theta}{dx}\left(1 + \left(\frac{dy}{dx}\right)^2\right)^{-\frac{1}{2}} \text{ since } \frac{ds}{dx} = \sqrt{1 + \left(\frac{dy}{dx}\right)^2}$$

To find $\frac{d\theta}{dx}$ only in terms of x, we can take advantage of the fact that the circle has the same tangents as the curve at the points where they touch. $\Delta\theta$ can be first found in terms of the normal vectors, $\boldsymbol{n_1}$ and $\boldsymbol{n_2}$, facing away from the circle at the points.

$$\Delta\theta = \arccos\left(\frac{\boldsymbol{n_1} \cdot \boldsymbol{n_2}}{|\boldsymbol{n_1}||\boldsymbol{n_2}|}\right)$$

Since the normal of a vector in two-dimensional space is found by simply swapping the x- and y-components and changing the sign of one of them, the angle between the corresponding tangent vectors with positive x-components will also equal to $\Delta\theta$. Therefore, this must also be true:

$$\Delta\theta = \left|\arctan\left(P'(x + \Delta x)\right) - \arctan\left(P'(x)\right)\right|,\ \Delta x > 0,$$

where $P'(x + \Delta x)$ and $P'(x)$ are the slopes of the tangents at the points.

Let $h(x)=\arctan\left(P'(x)\right)$

$$\frac{\Delta\theta}{\Delta x}=\frac{\left|h(x+\Delta x)-h(x)\right|}{\Delta x}$$

$$\Rightarrow \lim_{\Delta x\to 0}\frac{\Delta\theta}{\Delta x}=\left|\lim_{\Delta x\to 0}\frac{h(x+\Delta x)-h(x)}{\Delta x}\right|$$

$$\Rightarrow \quad \frac{d\theta}{dx}=\left|h'(x)\right|$$

$$=\left|\frac{d}{dx}\arctan\left(P'(x)\right)\right|$$

$$=\left|\frac{1}{1+P'(x)^2}P''(x)\right|$$

$$\therefore \kappa=\frac{\left|P''(x)\right|}{\left(1+P'(x)^2\right)^{1.5}} \tag{10}$$

The MATLAB function can now plug all the elements of x into equation (10) to obtain another set κ in which its $(j+1)$th element is found by using the $(j+1)$th element of x. All the elements in κ can then be plugged into the piecewise function f of r below, which was found earlier in equation (3), knowing that $\kappa=\frac{1}{r}$.

$$f(r)=\begin{cases}3.73\sqrt{r}+1.75, & 0\le r<30.5\\ 22.3, & r\ge 30.5\end{cases}$$

After the calculations, the outputs will be placed in another set $v=\{v_0, v_1, v_2, \ldots, v_{N-1}\}$ in the same manner as before. Using the sets v and s, I can finally produce a speed versus distance graph with line segments connecting the points (s_j, v_j), where s_j and v_j are the $(j+1)$th elements of s and v (see Figure 13).

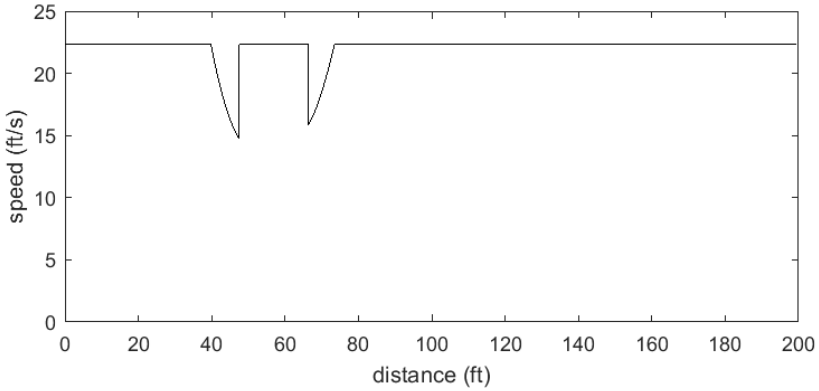


Fig. 13. A speed vs distance graph obtained by pairing up elements of the sets s and v and plotting the pairs with line segments connecting them.

However, I must not forget to consider my maximum and minimum accelerations. The curve in Figure 13 will be corrected using the following equation of motion:

$$v_f^2 = v_i^2 + 2a\Delta s$$

Because the curve in Figure 13 shows the greatest speed I could have at different distances depending on the curvature of a path, we are now going to lower, not raise, the curve as necessary to satisfy the acceleration constraints, which are that $a_{min} \le a \le a_{max} \Rightarrow -15.8 \le a \le 7.18$. To do that, the MATLAB function first sets $j = 1$ and $v_0 = v_{N-1} = 0$ (the latter equation must always stay true). Then, it checks whether the maximum acceleration constraint is unsatisfied, that is

$$v_j > \sqrt{v_{j-1}^2 + 2a_{max}(s_j - s_{j-1})}.$$

If the above inequality is true, the value of v_j will be decreased to equal $\sqrt{v_{j-1}^2 + 2a_{max}(s_j - s_{j-1})}$. If it is false, we will check whether the minimum acceleration constraint is unsatisfied, that is

$$v_j < \sqrt{v_{j-1}^2 + 2a_{min}(s_j - s_{j-1})} \Rightarrow v_{j-1} > \sqrt{v_j^2 - 2a_{min}(s_j - s_{j-1})}.$$

If it is, the value of v_{j-1} will be decreased to equal $\sqrt{v_j^2 - 2a_{min}(s_j - s_{j-1})}$. Notice that I am not simply increasing v_j because, if I do, I run the risk of having speeds that are too fast for the curvature of a path. However, we do get a problem nonetheless; decreasing v_{j-1} may result in v_{j-2} not satisfying the minimum acceleration constraint anymore. Therefore, if v_{j-1} does not satisfy, decrease its value and check if v_{j-2} satisfies. If v_{j-2} does not satisfy, change its value and check if v_{j-3} satisfies, and so on until a speed is found to satisfy the minimum acceleration constraint without being manipulated.

After all the acceleration constraints are satisfied, the MATLAB function increments j by 1 and repeats the checks and corrections. When it gets to the point where $j = N$, the MATLAB function stops incrementing, and the curve in Figure 14 is produced.

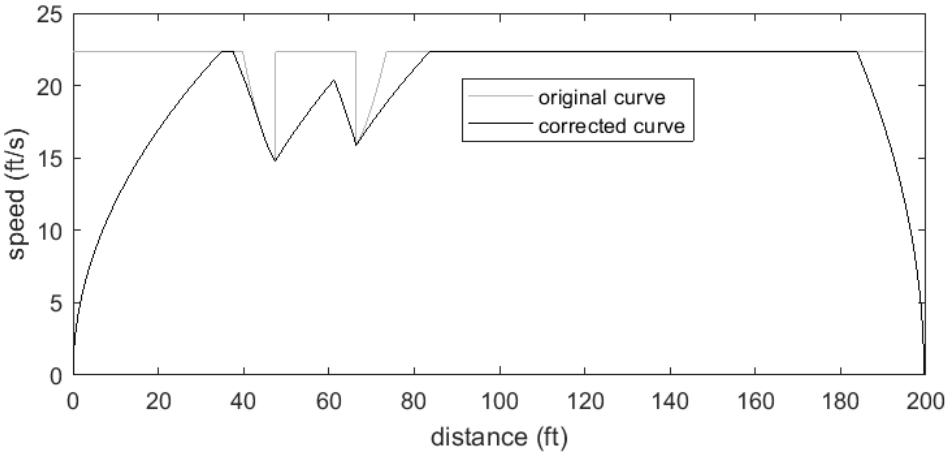


Fig. 14. A speed vs distance graph of my approximated sprint from home to second base along the path produced in this iteration.

3.4. Calculating Time

Now that I have a realistic speed versus distance graph, all I need to do is calculate the total time t it would take for me to run along the path in this iteration. To do that, according to the equation $t=\int \frac{1}{v}ds$ that was found earlier, we need to take the reciprocal of the equation that defines the curve in Figure 14 and integrate it with respect to s. Since the curve in Figure 14 is a series of connected line segments, the time Δt_j for an interval $\left[s_{j-1}, s_j\right[$ is calculated as follows:

$$\Delta t_j = \int_{s_{j-1}}^{s_j} (m_j s + b_j)^{-1}\, ds,$$

where $m_j = \dfrac{v_j - v_{j-1}}{s_j - s_{j-1}}$, and b_j is a constant such that $v_j = m_j s_j + b_j$ and $v_{j-1} = m_j s_{j-1} + b_j$.

For $m_j \neq 0$:

$$\begin{aligned}\Delta t_j &= \left[\frac{1}{m_j}\ln\left|m_j s + b_j\right|\right]_{s_{j-1}}^{s_j} \\ &= \frac{1}{m_j}\left(\ln\left|m_j s_j + b_j\right| - \ln\left|m_j s_{j-1} + b_j\right|\right) \\ &= \frac{1}{m_j}\left(\ln v_j - \ln v_{j-1}\right) \\ &= \frac{1}{m_j}\ln\frac{v_j}{v_{j-1}}\end{aligned}$$

For $m_j = 0$:

$$\begin{aligned}\Delta t_j &= \left[\frac{1}{b_j}s\right]_{s_{j-1}}^{s_j} \\ &= \frac{1}{b_j}\left(s_j - s_{j-1}\right) \\ &= \frac{1}{v_j}\left(s_j - s_{j-1}\right)\end{aligned}$$

$$\therefore \Delta t_j = \begin{cases} \dfrac{1}{m_j}\ln\dfrac{v_j}{v_{j-1}}, & m_j \neq 0 \\ \dfrac{1}{v_j}\left(s_j - s_{j-1}\right), & m_j = 0 \end{cases}$$

$$\therefore t = \sum_{j=2}^{N-2} \Delta t_j \tag{11}$$

Notice that the summation in equation (11) excludes the first and last segments of the curve in Figure 14. This was done because $v_0 = v_{N-1} = 0$, which means that Δt_0 and Δt_{N-1} are undefined. The omission will not affect the best path we will receive from the MATLAB function because all paths will let me run exactly at the same speed for the first and last $0.06\,\text{ft}$, especially when the size of the intervals $[s_0, s_1]$ and $[s_{N-2}, s_{N-1}]$ is exactly $0.06\,\text{ft}$. That is why I made $s_1 = 0.06\,\text{ft}$ and $s_{N-2} = s_{N-1} - 0.06\,\text{ft}$.

In this iteration, the time I would take to run along the path was calculated to be $11.43\,\text{s}$. If this time is the best so far, the value and the path will be stored in the MATLAB function until it finds a better path. Otherwise, the path produced in this iteration will be disregarded. The MATLAB function will repeat the process of producing a path, plotting a speed versus distance graph, and calculating time for the chosen number of iterations.

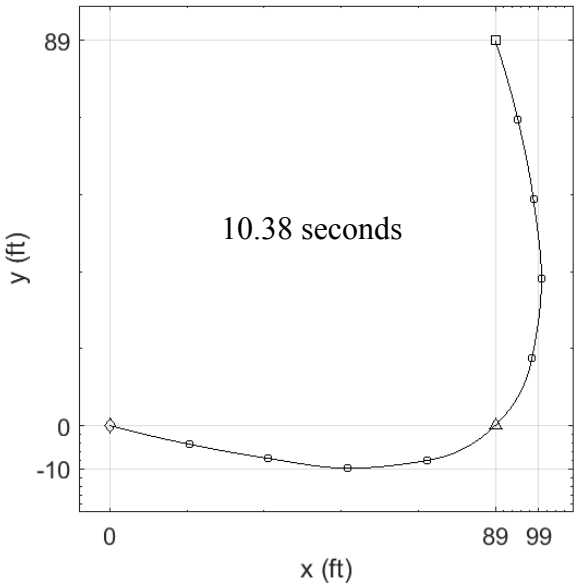
Finally, I am going to discuss how I found to use the step length of $0.06\,\text{ft}$. The way I did it was by finding the time difference of two random paths using a small step length of $0.01\,\text{ft}$, comparing it with the time difference of the same two paths but found with a larger manipulated

step length, and repeating the prior steps with different paths and a decreasing manipulated step length. The manipulated step length was initially 1 ft and would decrease by 0.01 ft after an undesirable event, which was when the differences have opposite signs and the modulus of the more precisely calculated difference is greater than 0.01 s at the same time. This situation is undesirable because, assuming that the more precisely calculated difference has the same sign as the exact difference, a difference calculated using a manipulated step length with an opposite sign means that the step length mistakes paths as better or worst than others when they are not supposed to be. However, if the modulus of the exact difference is less than or around 0.01 seconds, it would not matter anymore how they order themselves from best path to worst. It would not matter because, while collecting my data, the hundredth decimal place for time was consistently the uncertain digit, and thus, a path that allows me to run at most 0.01 s faster is just as good as a path that does not.

After running the small-scale algorithm for around 10,000 iterations, the step length decreased to 0.06 ft. I ran it for that many iterations because the step length was at 0.06 ft and did not decrease even after 1,000 iterations. Therefore, an approximation of the optimum path will be found using a step length of 0.06 ft.

3.5. Finding the Optimal Path

Now that the MATLAB function works properly, I ran it once for 100,000 iterations and a second time for 2,000,000 iteration, and the following approximated optimal paths were produced (see Figures 15, 16, 17, and 18).



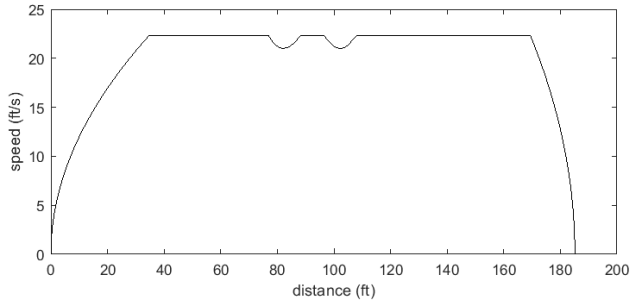
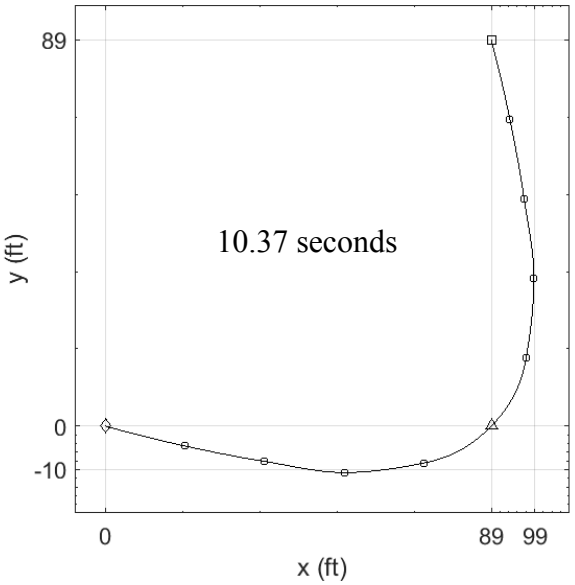


Fig. 15. A speed vs distance graph of a sprint on an approximate optimum path produced after 100,000 iterations from home to second base.

Fig. 16. An approximately optimal path and its time found after producing 100,000 paths.



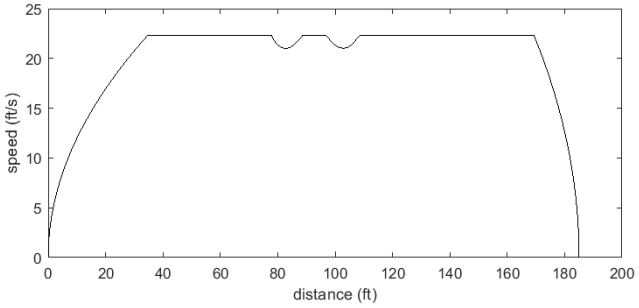


Fig. 17. A speed vs distance graph of a sprint on an approximate optimum path produced after 2,000,000 iterations from home to second base.

Fig. 18. An approximately optimal path and its time found after producing 2,000,000 paths.

From the produced paths after 100,000 and 2,000,000 iterations, we can conclude that the optimum path is approximately symmetric, goes off the baseline by around $10\,\text{ft}$ at most, and starts with a tangent approximately making a 14º angle with the baseline. Although it seems that the dips in both speed versus distance graphs only increase time, they also allow the paths to be closer to the baseline, which means that their arc lengths are on the shorter side. This characteristic reduces the calculated times very well. Therefore, to minimize the time to sprint from home to second base, I must slow down a little bit at first base in an effort to decrease the distance I travel. Note that the paths are based on the collected data of my sprints, which brings up the question, what if I collect data from somebody else's sprints? Would the resulting path change? If it does, by how much? My hypothesis is that the resulting path would be different from mine and the maximum running speed and acceleration have very little effect on the optimal path, while the dataset for circular sprints makes a large impact. This is because if the piecewise function f of r initially grows faster than it does for me, the better paths will be able to sacrifice small curvature for shorter arc lengths, which means that the optimal path will be shorter and closer to the baseline. On the other hand, if maximum speed and acceleration increase, the MATLAB function will just be working with the same dips but taller graphs, which means that time would mostly be affected rather than the optimal path.

Although they look very optimized, the results from this exploration cannot be fully trusted because, without the restriction on right turns, there would be $31^8 \approx 8.53 \times 10^{11}$ (the number of nodes in each column to the power of the number of columns) different paths that can be produced, and thus, even with the restriction, the path produced after 2,000,000 iterations is most likely not the best path that the MATLAB function can produce with the collection nodes. However, the fact that it is very similar to the path produced after 100,000 iterations allows conclusions to be made about the optimum path that many, if not most, baseball players do not know of.

The best approximation of the optimal path shown in Figure 18 has a shape similar to the one found by the one article[1] that optimized a base runner's path from home to second base (see Figure 19). However, the path from this exploration is less rounded, and the path from the article starts by going more than 20º right of the baseline instead of around 14º. This is most likely due to the fact that their path is a general one for anyone who wants to get to second base in the

shortest time possible while the path in Figure 18 is found based on how I would approximately run along paths given my physical abilities. And it is not just my abilities that affect the optimal path either; the conditions of the baseball field I collected my data on matters, too, and thus, optimal paths can vary even when focusing on a single individual. As a result, comparing does not help to determine the validity of my results. I would need to time myself on the same baseball field I collected my data on in order to see if they are reliable.

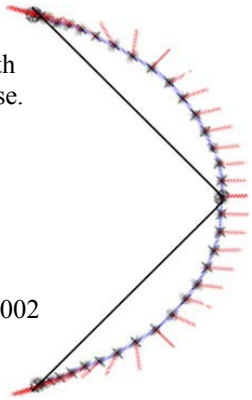
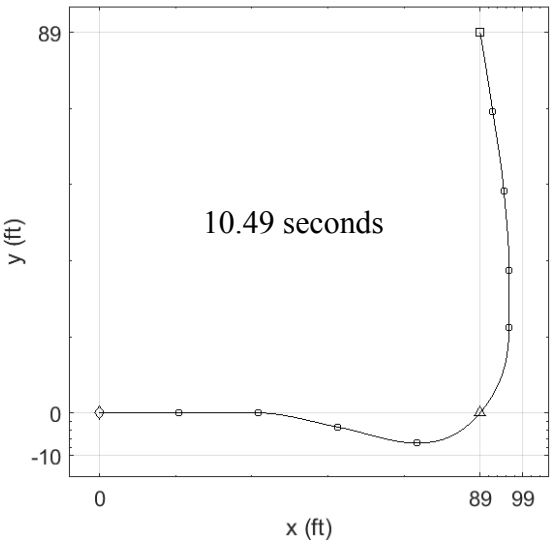


Fig. 19. The fastest path from home to second base. Carozza, D., Johnson, S. and Morgan, F. "Baserunner's Optimal Path." Math Intelligencer 32, no. 4 (2010): 10-15. https://doi.org/10.1007/s00283-009-9106-2.

To see how my approximated optimum compares with the path that I had always been taught to use, compare the previous figures with Figure 20 and 21. As expected, the path is suboptimal.



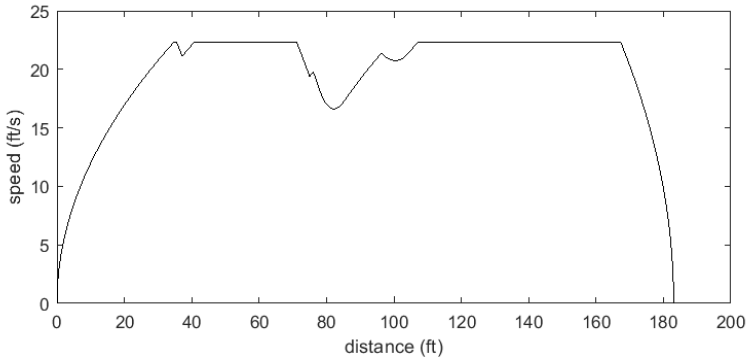


Fig. 20. A speed vs distance graph of a sprint on the path I had been taught to use from home to second base.

Fig. 21. A path with a shape similar to the one I had been taught to use, and its time.

4. Conclusion

In my Math Exploration, I wrote a MATLAB function that finds an approximate optimum path that gets me from home to second base on a baseball field. It produced many possibly-optimum paths and their corresponding speed versus distance graphs. For each path, the area under a curve after a reciprocal transformation was calculated to get the total length of time I would spend on it. The calculated times were constantly being compared to find the best path, which was displayed in the end.

After comparing the approximated optimum and the path that I was taught to use, I learned that the path I had been taking to second base, just like many other baseball players I know, for many years was slowing me down. Every time I wanted to reach a base beyond first, I should have started preparing for the turn at first base right at the beginning of my sprint, not halfway between home and first base. When I sprinted to second from home base, I should have been around 10 feet off the baseline two thirds of the way to first and one third of the way to second. Although I am not playing baseball anymore, I hope to help those who still are, such as my former teammates, improve their performance on the field. The MATLAB function will find

their own optimal running paths just like it did for me. All they would need to do is collect data for themselves according to my methods, change a few lines of code in the MATLAB function, and run it. Once they see their simulated fastest path, I believe that they will greatly benefit from it. Even if the path that they have been taking to second base is almost as good as the approximated optimum, a small adjustment may end up making a big difference to their performance and team's outcome of a game.

This MATLAB function for finding the optimal path can be applied not only when a runner is at home base, but also when the runner is trying to advance two or more bases in general. Furthermore, it can be applied to other sports, such as car racing and any other requiring something to reach somewhere as fast as possible using a unique path, and even to the brachistochrone problem, even though it would not be the best way to solve it.

The possible limitations to this exploration were as follows:

- The equation to fit the data points in the speed versus radius graph may not be an accurate model of the real physics, which might be much more complicated than it seems. There are also many uncontrolled variables: my average speed might have been lower than it should have been for the larger circles due to my limited endurance; the ground I was running on may not have been flat; and I may have been running on dirt that varies in slipperiness to a significant amount.
- The second derivative of the PCHIP functions used is said to be most likely not continuous[7]. This may have caused problems because the curvature formula makes use of the second derivative. It is probably the reason why Figure 13 has jumps in speed even though its corresponding path was very smooth.
- I only found one article[1] with the purpose of finding the best path one could take from home to second base, and it used a method I could not understand. Therefore, I referred to an article[3] optimizing car racing lines instead. I understood this article, but it only explained the general idea of their methods. In the end, I only used it for inspiration to come up with my own simple method of finding an approximation of the optimal path.

The following ideas are based on the racing line article[3]: to improve the results in this IA, instead of having discrete nodes spanning the width of 30.5 ft, I could make use of continuous vertical lines for each column instead of nodes. Then, I would make certain segments of the lines more likely to be picked depending on the previous choice of point using probability distributions. This method will take care of the smoothing aspect we had dealt with earlier due to the continuity of the lines. I would also code the MATLAB function so that every time it finds a new path, the probability distributions shift to favour that path. This method would get a very precise result due to not using discrete nodes anymore, and I think it would take a reasonable amount of time to find a good approximate optimum due to the use of probability distributions.

Appendix

The continuous dataset collected to find my maximum running speed and my maximum and minimum accelerations

Time (s)	Distance (in)	Time (s)	Distance (in)
0	0	5.18	1146.0
0.46	32.5	5.43	1213.0
0.73	69.0	5.68	1285.0
0.97	107.5	5.92	1350.0
1.20	155.5	6.18	1417.5
1.43	203.0	6.42	1484.5
1.67	256.5	6.65	1555.0
1.90	307.5	6.92	1622.5
2.13	365.5	7.16	1694.0
2.40	423.0	7.42	1762.0
2.64	484.5	7.69	1831.5
2.87	545.0	7.96	1900.0
3.11	611.0	8.22	1981.0
3.37	676.0	8.56	2063.5
3.64	743.5	8.83	2127.5
3.87	810.0	9.02	2157.0
4.14	881.5	9.23	2184.0
4.38	947.0	9.39	2198.5
4.64	1015.0	9.63	2198.5
4.91	1077.5	N/A	N/A

Works Cited

[1] Carozza, Davide, et al. *Baserunner's Optimal Path*, advised by Morgan Johnson, 2009, web.williams.edu/Mathematics/fmorgan/Baserunner.pdf.

[2] MATLAB 2018a, The MathWorks, Inc., Natick, Massachusetts, United States.

[3] Xiong, Ying. *Racing Line Optimization*, certified by Gilbert Strang, accepted by Karen Willcox, 2010, pp. 9, 54–62, dspace.mit.edu/bitstream/handle/1721.1/64669/706825301-MIT.pdf.

[4] Tom O'Haver (2020). Fast smoothing function (https://www.mathworks.com/matlabcentral/fileexchange/19998-fast-smoothing-function), MATLAB Central File Exchange. Retrieved January 25, 2020.

[5] Dawkins, Paul. "Arc Length". *Paul's Online Notes*, 2003–2020, tutorial.math.lamar.edu/Classes/CalcII/ArcLength.aspx.

[6] "Deriving the Curvature Formula". *Youtube*, uploaded by Awol Geordie, 13 February 2017, www.youtube.com/watch?v=uaCAILIo6Ls.

[7] "pchip". *MathWorks*, 1994–2020, www.mathworks.com/help/matlab/ref/pchip.html.