

# The Fountain that Math Built

Alex McCauley, Josh Michener, and Jadrian Miles  
Dr. Dan Teague, NCSSM, 2002

## **Abstract**

We are presented with a fountain in the center of a large plaza, which we wish to be as attractive as possible but to not splash passersby on windy days. Our task is to design an algorithm that controls the flow rate of the fountain, given input from a nearby anemometer. During calm winds, the fountain sprays out water at a steady rate. Whenever the wind picks up sufficiently, the flow is attenuated, so as to keep the water within the fountain's pool; in this way, we strike a balance between aesthetics and comfort.

We considered the water stream from the fountain as a collection of different-sized droplets which initially leave the fountain nozzle in the shape of a perfect cylinder. This cylinder is then broken into its component droplets by the wind, with the smaller droplets being carried farther than the larger. In the reference frame of the air, a droplet is moving through stationary air and experiencing a drag force as a result; since the air is moving with a constant velocity relative to the fountain, the force on the droplet is the same in either frame of reference.

Modeling this interaction as laminar flow, we arrived at equations for the drag forces. From these equations, we derived the acceleration of the droplet, which we integrated to find the equations of motion for the droplet. These allow us to find the time when the droplet hits the ground from the height equation and, assuming that it lands at the very edge of the pool, the time when it reaches its maximum range from the horizontal position equation. Equating these and solving the initial flow rate, we arrived at an equation for the optimal flow rate at a given constant wind speed. Since the wind speeds will not be constant, the algorithm must make its best prediction of wind speed and use current and previous wind speed measurements to damp out transient variations.

Our final solution is an algorithm which takes as its input a series of wind speed measurements and determines in real-time the optimal flow rate to maximize the attractiveness of the fountain while avoiding splashing passersby excessively. Each iteration, it adds an inputted wind speed to a buffer of previous measurements. If the wind speed is increasing sufficiently, the last .5 seconds of the buffer are considered; otherwise, the last 1 second is. The algorithm computes a weighted average of these wind speeds, weighting the most recent value slightly more than the oldest value considered. It then uses this weighted velocity average in the equation which predicts the optimal flow rate under constant wind. The result is the optimal flow rate under variable wind, knowing only current and previous wind speeds.

## **Introduction**

We are presented with a fountain in the center of a large plaza, which we wish to be as attractive as possible but to not splash passersby on windy days. Our task is to design an algorithm that controls the flow rate of the fountain, given input from a nearby anemometer. During calm winds, the fountain sprays out water at a steady rate. Whenever the wind picks up sufficiently, the flow is attenuated, so as to keep the water within the fountain's pool; in this way, we strike a balance between aesthetics and comfort.

## **Variables**

A list of relevant variables, constants, and parameters is included as Appendix 1.

## **Assumptions**

We made several simplifying assumptions:

- Passersby find a higher spray more attractive.
- Avoiding discomfort is more important to passersby than the attractiveness of the fountain.
- The water stream can be considered a collection of spherical droplets, each of which has no initial horizontal component of velocity.
- Every possible size of sufficiently small water droplet is represented in the water stream in significant numbers.
- Water droplets remain spherical.
- The interaction between the water droplets and wind can be described as non-turbulent, or 'laminar' flow.
- There exists a minimum uncomfortable water droplet size; passersby find it acceptable to be hit by any droplets below this size but by none above.
- When the wind enters the plaza, its velocity is entirely horizontal.
- The wind speed is the same throughout the plaza at any given time.
- The pool and the area around it are radially symmetric, so there is no preferred radial direction.
- We can neglect any buoyant force on the water due to the air, since the error introduced by this approximation is equal to the ratio of densities of the fluids involved, on the order of  $10^{-3}$ , which is negligible.
- The anemometer reports wind speeds at discrete time intervals  $dt$ .

## **Analysis of the Problem**

If the water stream is viewed as a collection of small water droplets blown from a core stream, then the interaction between the droplets and the air moving past them can best be described in the inertial reference frame of the moving air. In this frame, the air is stationary while the droplet moves horizontally through the air with a speed equal to the relative speed of the droplet and wind,  $v_r = v_a - v_x$ . In the vertical direction,  $v_r = v_y$ , since the wind only blows horizontally.

In the air's frame of reference, the water droplet experiences a drag force opposing  $v_r$ . Assuming that the air moves at a constant velocity, this force is the same in both frames of reference. In the frame of the fountain, then, the droplet is being blown in the direction of the wind. The smaller water droplets are carried farther, so we need only consider the motion of the smallest uncomfortable water droplets, knowing that bigger droplets will not travel as far.

The water droplet initially has a vertical velocity,  $v_y(0)$ , which is directly related to the flow rate of water through the nozzle of the fountain. Thus, this initial vertical velocity component can be controlled by changing the flow rate. This motion will cause vertical air resistance, slowing the droplet and affecting the time the droplet spends in the air.

Since the vertical and horizontal components of a water droplet's motion are completely independent of each other, the time the water droplet spends in the air,  $t_w$ , is solely determined by the droplet's vertical motion. Knowing this time allows us to find the horizontal distance traveled by the water droplet, which we wish to constrain to the radius of the pool.

When the wind is variable, however, we cannot determine exactly what the ideal flow rate will be at any given time. We must instead act on our current reading and rely on previous measurements of wind speed in order to restrain the model from reacting too severely to wind fluctuations. We wish to react faster to increases in wind speed, since they result in splashing which is weighted more heavily.

### **Design of the Model**

For our initial model we assume that  $v_a$  is constant for time intervals on the order of  $t_w$ , so that any given droplet experiences a constant wind speed while in the air.

We model the water stream as a collection of droplets which are initially cohesive but are carried away at varying velocities by the wind. The distance these droplets are carried depends upon the wind speed  $v_a$  and the initial vertical velocity of the water stream through the nozzle,  $v_y(0)$ . Since the amount of water flowing through the nozzle per unit time is  $f = v_y(0) \cdot A$ ,  $v_y(0) = f/A$ . The dynamics of the system, then, are fully determined by  $f$  and  $v_a$ . First, we find the equations of motion for the droplet.

#### *Equations of Motion for a Droplet*

For laminar flow, a spherical particle of radius  $r$  traveling with velocity  $v$  through a fluid medium of viscosity  $\eta$  experiences a drag force,  $F_D$ , such that

$$F_D = (6\pi\eta r)v \quad [\text{Winters, 2002}]$$

Since a spherical water droplet has a mass given by

$$m = \rho_w \left( \frac{4}{3} \pi r^3 \right)$$

the acceleration felt by the droplet is given by Newton's Second Law as the total force over mass. Since there are no other forces acting in the horizontal direction, the horizontal acceleration  $a_x$  is given by:

$$a_x(t) = \frac{d^2x}{dt^2} = \left( \frac{9\eta_a}{2\rho_w r^2} \right) v_r = k(v_a - v_x) \quad \text{where } k \equiv \frac{9\eta_a}{2\rho_w r^2}$$

The droplet experiences both air drag and gravity in the vertical direction, so the vertical acceleration is:

$$a_y(t) = -\left[\left(\frac{9\eta_a}{2\rho_w r^2}\right)v_y + g\right] = -k\left(v_y + \frac{g}{k}\right)$$

With constant  $v_a$ , we use separation of variables and integrate these equations to find  $v_x(t)$  and  $v_y(t)$ , using the facts that  $v_x(0) = 0$  and  $v_y(0) = f/A$ . The results are:

$$v_x(t) = v_a(1 - e^{-kt})$$

$$v_y(t) = ne^{-kt} - \frac{g}{k} \quad \text{where } n \equiv \frac{g}{k} + \frac{f}{A}$$

Integrating again, and using  $x(0) = y(0) = 0$ , we have:

$$x(t) = \frac{v_a}{k}(kt + e^{-kt} - 1)$$

$$y(t) = \frac{1}{k}\left[n(1 - e^{-kt}) - gt\right]$$

### Determining $f$

Because  $f$  is the only parameter the algorithm modifies, we wish to find the flow rate that would restrict the smallest uncomfortable water droplets to ranges within  $R_p$ , so that they would land in the fountain's pool.

After a time  $t_w$ , the droplet has reached its maximum height and then fallen back to the ground. Thus,  $y(t_w) = 0$ . This equation is too difficult to solve exactly, so we use the expansion for  $e^{-kt}$ . Truncating after the quadratic term, we approximate the  $e^{-kt}$  term in  $y(t)$  using

$$e^{-x} \cong 1 - x + \frac{x^2}{2}. \quad \text{Solving } y(t_w) = 0, \text{ we find}$$

$$t_w \cong \frac{2}{k}\left(1 - \frac{g}{nk}\right)$$

We know that the maximum horizontal distance,  $x(t_w)$ , must be less than or equal to  $R_p$ , with equality holding for the smallest uncomfortable droplet. In this case,  $x(t_w) = R_p$ , and using the same expansion for  $e^{-kt}$  as used above,

$$R_p = x(t_w) \cong \frac{v_a}{k}\left(kt_w - 1 + 1 - kt_w + \frac{(kt_w)^2}{2}\right) = \frac{v_a k}{2} t_w^2$$

Solving for  $t_w$  and equating it to the earlier expression for  $t_w$ ,

$$\sqrt{\frac{2R_p}{v_a k}} = t_w = \frac{2}{k}\left(1 - \frac{g}{nk}\right)$$

Recalling that in this equality only  $n$  is a function of  $f$ , we substitute for  $n$  and solve for  $f$ . The result is:

$$f(v_a) = \frac{Ag}{\sqrt{\frac{2v_a k}{R_p} - k}} \quad (1)$$

As  $v_a \rightarrow \frac{kR_p}{2}$ , equation 1 becomes singular (see page 6, Figure 2). At lower values of  $v_a$ , equation 1 gives a negative flow rate. These wind speeds are very small; at such speeds, the droplets would not be significantly deflected by the wind. Equation 1 assumes that flow rate can be made arbitrarily high, and therefore is unrealistic and invalid in application. To make the model more reasonable, then, we modify this equation to include the maximum flow rate achievable by the pump,  $f_{\max}$ :

$$f(v_a) = \min \left( \frac{Ag}{\sqrt{\frac{2v_a k}{R_p} - k}}, f_{\max} \right) \quad v_a > \frac{kR_p}{2}$$

$$f(v_a) = f_{\max} \quad v_a \leq \frac{kR_p}{2} \quad (2)$$

An algorithm can use the given constants and a suitable minimal droplet size to determine the appropriate flow rate for a measured  $v_a$ . However, as previously stated, equation 2 assumes that the wind speed is constant over the time scale  $t_w$  for any given droplet. A more realistic model must take into account variable wind speed.

### *Variable Wind Speeds*

When wind speed is allowed to vary with time, the physical reasoning used above becomes invalid since the relative velocity of the reference frames is no longer constant. Mathematically, this is manifested by the equation for velocity-dependent horizontal acceleration. Integrating this equation is now not so simple, and we must resort to numerical means to find the equations of motion. Additionally, the algorithm has no knowledge of future wind fluctuations, and can only rely on past and present wind data to find the appropriate flow rate. Our model needs to incorporate these wind data to make a reasonable prediction of the wind's velocity over the next  $t_w$  and determine an appropriate flow rate using equation 2.

A gust is defined to be a sudden wind speed increase on the order 5 m/s that lasts for no more than twenty seconds; a squall is a similarly sudden wind speed increase that lasts longer than a gust [Weather.com, 2002]. Our model should account for gusts and squalls, and for “reverse” gusts and squalls, in which the wind speed suddenly decreases. Since wind speeds can change drastically and unpredictably over flight time of the droplet, our model will behave badly at times and there is no way to completely avoid this, only to minimize its effects.

Our reaction to wind speed is not fully manifested until the droplet lands, after a time  $t_w$  (approximately two seconds). By the time our model has reacted to a gust or reverse gust, therefore, the wind speed has stopped changing. Without some type of buffer, in a gust our model would react by suddenly dropping flow rate as the wind speed peaked and then increasing it again as the wind speed decreased; the fountain would virtually cut off for the duration of any gust, which would release less water and thus seem very unattractive to passersby. Additionally, the water released just before the onset of the gust would be airborne as the wind speed picked up, splashing passersby regardless of any reaction by our model.

In a reverse gust, without a buffer, the flow rate would increase suddenly, so water released at that time would be caught by the wind as its speed picked up: the fountain would release an attractive jet of water, but passersby would get splashed.

We devised an algorithm for analyzing wind data that makes use of equation 2. Because velocity now varies within times of  $t_w$ , we do not want to directly input the current wind speed, but rather a buffered value, so that our model does not react too sharply to transient wind changes. We want our model to react more quickly to sudden increases in wind speed than to decreases, because increases cause splashing, which we weight more heavily than attractiveness.

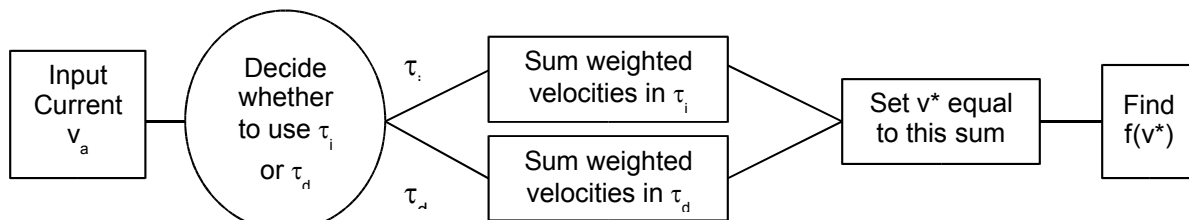
Our model, therefore, has two separate velocity buffer times: one,  $\tau_d$ , which is used by default and another,  $\tau_i$ , which is used when the wind speed increases drastically. We also weight the more recent values in the buffer more heavily, since we want our model to react fairly promptly to wind speed changes, but not overreact. We weight each value in the velocity buffer with a constant value,  $K$ , plus a weight proportional its age: less recent velocities are considered but given less weight than more recent ones. The weight of the oldest value in the buffer is  $K$ , and that of the most recent is  $K + 1$ , with a linear increase between the two. With the constraint that the weights are normalized (i.e., they sum to 1), the equation for the  $i^{\text{th}}$  weight factor is:

$$w_i = \frac{\left( K + \frac{i}{\tau - dt} \right) dt}{\left( K + \frac{1}{2} \right) \tau}$$

The velocities are multiplied by their respective normalized weights and then summed. This sum,  $v^*$ , is then used in equation 2 to find the appropriate flow rate for the fountain at a given time.

We wish our algorithm to use  $\tau_i$  rather than  $\tau_d$  when the wind speed increases sufficiently over a recent interval, but not when it increases slightly or fluctuates rapidly, such as for random variations in wind speed. We choose to satisfy this criterion by switching from  $\tau_d$  to  $\tau_i$  whenever the wind speed increases over two successive .2s intervals and by a total of at least 1 m/s over the entire .4s interval.

Our algorithm follows the following flow chart in computing the current flow rate:



**Figure 1.** Flow chart for computing flow rate with variable wind speed.

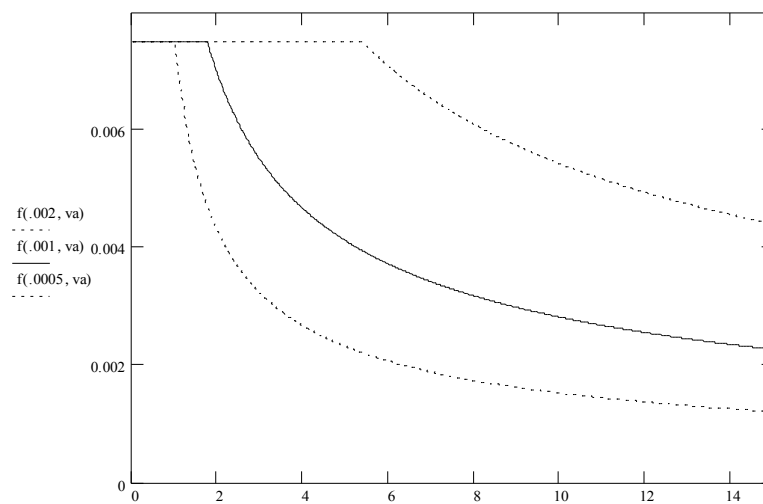
We wrote a C++ program to compute this algorithm, the code for which is included as Appendix 2.

## Testing and Sensitivity Analysis

### *Sensitivity of Flow Equation*

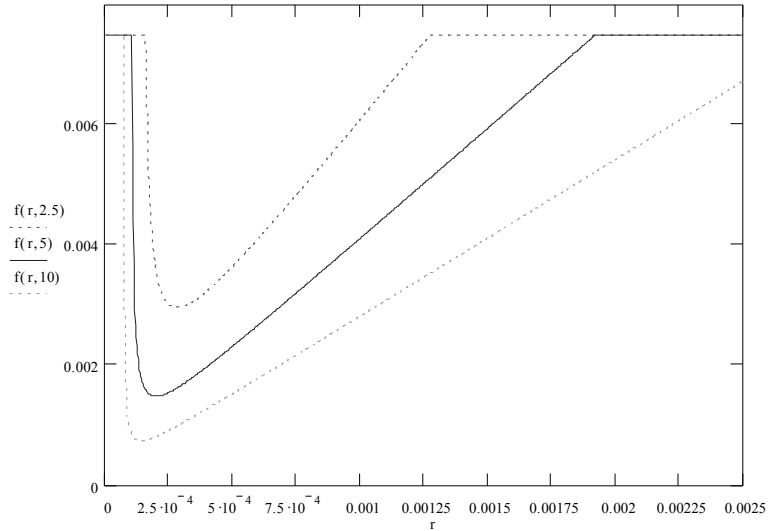
In our equation for flow rate, two variables can change: minimal droplet size and wind speed. While the minimal droplet size will not change dynamically, its value is a subjective choice that must be made by the owner of the fountain. The wind speed, however, will change dynamically throughout the problem, and the purpose of our model is to react to these changes.

We examined equation 2 for varying wind speeds and minimal drop sizes. The fountain used to generate the graphs below has a nozzle radius of 1 cm ( $A = \pi \cdot 10^{-4} \text{ m}^2$ ), a maximum flow rate of 7.5 L/s ( $.0075 \text{ m}^3/\text{s}$ ), and a pool radius of 1.2 m. This maximum flow rate is chosen for illustrative purposes and is not reasonable for such a small fountain.



**Figure 2.** Graphs of  $f$  vs.  $v_a$  for several values of  $r$ .

At any given wind speed, as the acceptable droplet radius decreases, the flow rate must also decrease. At higher wind speeds, this difference is less pronounced, but at lower speeds, the decision of acceptable size has a significant impact on the flow rate. At very low wind speeds, the fountain is not physically able to shoot the droplets high enough to allow the wind to carry them outside the pool, regardless of size. Our cutoff,  $f_{\max}$ , reflects that the fountain pump cannot generate the extreme flow needed to get the droplets to the edge of the pool in these conditions.

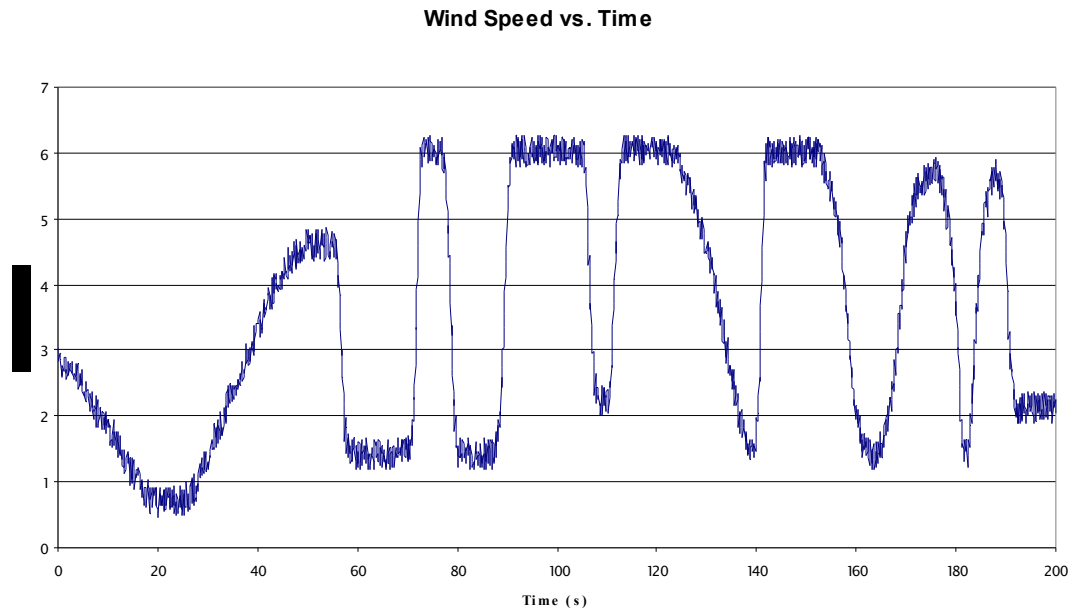


**Figure 3.** Graphs of  $f$  vs.  $r$  for several values of  $v_a$ .

At any given droplet size, as the wind speed increases, the flow rate must decrease in order to keep the droplets in the pool. For larger values of  $r$ , a change in wind speed requires a greater absolute change in flow rate than for smaller values of  $r$ . For very small droplets, the drag force dominates the force of gravity, and an increase in flow also increases the drag force to such an extent that the particle spends no more time in the air. This behavior is readily apparent in the equation for vertical acceleration on page 3 as  $r$  approaches zero. These extremely small values of  $r$ , though, describe droplets that are unlikely to discomfort passersby, and thus are not significant to our model.

#### *Sensitivity of Flow Algorithm*

The results of our algorithm depend on the parameters  $\tau_i$ ,  $\tau_d$ , and  $K$ , which determine the size of the buffer and weights of the velocities in the buffer. To test the sensitivity of our algorithm to these parameters and to find reasonable values for them, we created the following set of simulated wind speeds, including small random variations, on which to test our algorithm. This data set does not reflect typical wind patterns but includes a variety of extreme conditions.



**Figure 4.** 3-minute simulation of wind speeds.

We wished to create a quantitative estimate of the deviation from ideal performance of our flow algorithm and then test the algorithm with different combinations of parameters to find the set which produced the smallest deviation under the simulated wind conditions.

To quantitatively measure how ‘bad’ a set of flow choices are, we only considered the droplets which fall outside the pool, since those are the droplets which splash passersby. The ‘badness’ of the algorithm’s reaction, then, is the sum over the run of the simulation of the distances outside the pool at which droplets land.

To determine this distance, we needed to know how the droplets actually moved through the air in varying wind speeds. Describing this motion in closed form is mathematically impossible without continuous wind data, so we chose to approximate the equations of motion with an iterative process.

Since the time a particle spends in the air,  $t_w$ , is not affected by the wind speed, we knew  $t_w$  for each particle. We stepped through the time  $t_w$  in intervals of  $dt$ , computing the particle’s acceleration, velocity, and position as:

$$\begin{aligned} a_i &= k(v_{a,i} - v_i) & a_0 &= kv_{a,0} \\ v_i &= v_{i-1} + a_{i-1}dt & v_0 &= 0 \\ x_i &= x_{i-1} + v_i dt & x_0 &= 0 \end{aligned}$$

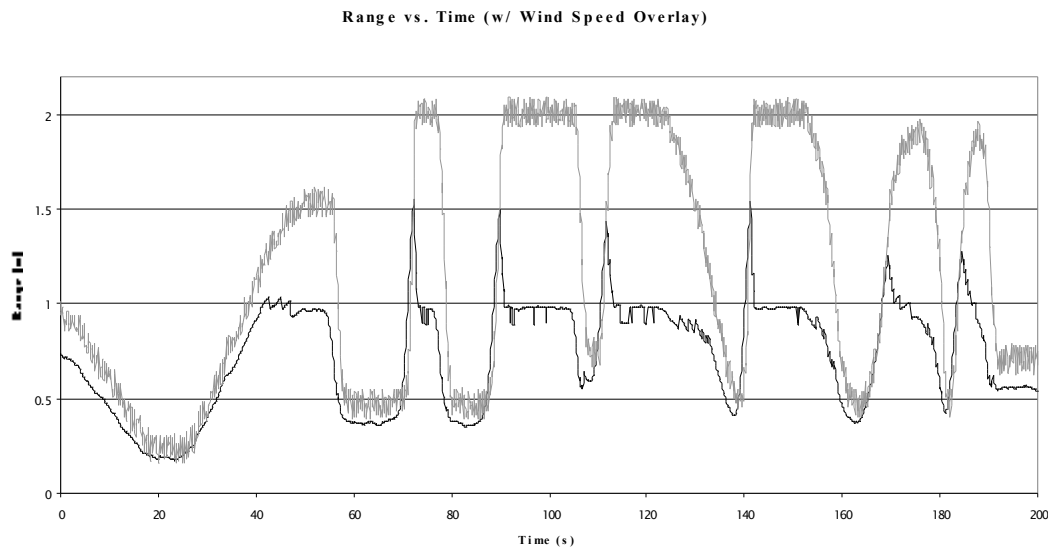
When we reach  $t_w$ , the droplet has hit the ground, and we compared its horizontal position to the radius of the pool. This was done for each droplet released during the simulation, keeping track of both the largest absolute difference and the average difference. We wrote a program to calculate these values for a given set of flow and wind data, outputting them to a data file. The code for this program is included as Appendix 3.

To test the flow algorithm with many combinations of parameters, we ran it with each combination, outputting the calculated flow rates to a data file. We then ran our test program on

each set of flow data, comparing the greatest and average deviations for each set of parameters. The set of parameter values that produced the least deviation were  $\tau_i = .5$ ,  $\tau_d = 1$ , and  $K = 10$ . These values imply that only fairly recent wind speed measurements should be held in the buffer, while the most recent velocity should have a weight of  $\frac{K+1}{K} = 1.1$  relative to the oldest.

Lowering  $K$  beyond this value increased the deviation from the ideal, while increasing it further made no difference. Similarly, increasing  $\tau_i$  or  $\tau_d$  increased the deviation because the algorithm could not quickly respond to changes in wind speed. Decreasing  $\tau_i$  below .5 made no difference, while decreasing  $\tau_d$  would make the model too sensitive to short fluctuations in wind speed.

**Figure 5.** Range of droplets over the simulation overlaid with scaled wind speeds.



## **Justification**

### *Validity of the Laminar Flow Assumption*

Our model was based on a drag force proportional to  $v_r$ ; this, however, is not necessarily correct. For higher speeds or large droplet sizes, the drag becomes proportional to  $v_r^2$ . We thus need to determine whether reasonable physical scenarios allow us to model the drag force as being proportional to  $v_r$  and not its square.

For a sphere of radius  $r$  moving through the air with velocity  $v_r$ , the Reynolds number  $R$  is defined to be

$$R = \frac{2\rho_a v_r r}{\eta_a} \quad [\text{Winters 2002}]$$

When  $R < 10^3$ , there is little turbulence and laminar flow dominates, so air resistance is roughly proportional to  $v_r$ . If  $R > 10^3$ , the flow is turbulent, in which case the drag force is proportional to  $v^2$  [Winters 2002]. Thus, for our assumption of laminar flow to hold, our calculated Reynolds number must be less than  $10^3$ . Plugging in values for the constants above, and using a physically reasonable relative velocity of 4.5 m/s (corresponding to a wind speed of roughly 10 miles per hour), we obtain  $R = (5.8 \times 10^5) \cdot r$ , which satisfies the given constraints for

predominantly laminar flow when  $r < 1.7$  mm. Because water droplets of diameter  $> 3$  mm are uncomfortable, these provide an upper limit on the possible droplet sizes to be considered in the model.

Because these smaller droplets will bound the larger droplets in how far they go from the fountain (see below), then all of our analysis is concerned with droplets whose sizes are within the allowed range for laminar flow.

### *Bounding the Droplet Range*

For either laminar or turbulent flow, the acceleration due to drag scales with as  $F/m \propto r^{-n}$ , where  $1 \leq n \leq 2$ . Larger droplets therefore experience a lower horizontal acceleration due to drag, while acceleration in the vertical direction is dominated by gravity ( $k < .1g$ ), so that the time a particle spends in the air is roughly the same for droplets of varying radius. The heavier droplets have less horizontal acceleration, so they travel a shorter horizontal distance in the same amount of time than smaller droplets. The ranges are, therefore, shorter for larger droplets, so we can bound all uncomfortably-sized droplets by the range of the smallest such droplet.

### *Initial Shape of the Water Stream*

We assumed that the water coming out of the fountain nozzle would have no initial horizontal velocity; that is, the stream would be a perfect cylinder with the same radius as the nozzle. In fact, the stream would be closer to the shape of a steep cone, and the droplets would have some horizontal velocity. In the absence of wind, this would have a significant impact on where the droplets landed, since without wind our algorithm predicts a horizontal range of zero. However, in these cases, the flow rate would be bounded by  $f_{\max}$  regardless of initial velocity, so the natural spread of the fountain is irrelevant. In higher wind, the initial horizontal velocity is quickly dominated by the acceleration due to the wind and thus makes a negligible contribution to the total range.

### *Exclusively Horizontal Wind*

We assumed that the wind would be exclusively horizontal. Since the anemometer measures only horizontal wind speed, it is the only component we can consider in our model. Additionally, the buildings around the plaza would tend to act as a wind tunnel and channel the wind horizontally.

### *Quadratic Approximation of $e^{-kt}$*

Because the series for  $e^{-kt}$  is alternating, the error from truncating after the second term is no greater than the third term, which is  $\frac{(kt)^3}{6}$ . The relative error is  $\frac{(kt)^3}{6e^{-kt}} \approx .001$  for reasonable values of  $k$  and  $t$ , so our approximation introduces very little error.

## **Conclusions**

Our final solution is an algorithm which takes as its input a series of wind speed measurements and determines in real-time the optimal flow rate to maximize the attractiveness of the fountain while avoiding splashing passersby excessively. It takes an inputted wind speed and adds it to a buffer of previous measurements. If the wind speed is increasing sufficiently, the last .5 seconds of the buffer are considered; otherwise, the last 1 second is. The algorithm computes a weighted average of these wind speeds, weighting the most recent value 10% more heavily than the oldest value considered. It then takes this weighted average and uses it in the equation that predicts the optimal flow rate under constant wind. The result is the optimal flow rate under variable wind, knowing only current and previous wind speeds.

### **Strengths and Weaknesses**

#### *Strengths*

- Given reasonable values for the characteristics of the fountain and for wind behavior, our model returns values that satisfy our goal of maintaining an attractive fountain without excessively splashing passersby.
- The model can compute optimal flow rates in real time. Running one cycle of the algorithm takes a time on the order of .001 seconds, so the fountain's pump could be adjusted as fast as physically possible.
- The constants which determine the behavior of the algorithm,  $\tau_i$ ,  $\tau_d$ , and  $K$ , are not arbitrary but instead are those which performed best under simulation.
- Our algorithm is very robust; it works well under extreme conditions and can be readily modified for different situations or fountains.

#### *Weaknesses*

- One of our primary assumptions is that the droplets coming from the fountain nozzle have no horizontal velocity. In reality, the nozzle sprays a cone of water, rather than a perfect cylinder as we assumed. As explained above, this does not have a significant impact on the results of the algorithm.
- Another important assumption of our model is the use of laminar flow. The water droplets are of a size to experience a combination of laminar and turbulent flow, but describing such a combination of regimes is mathematically difficult and is only known through experimentation. A more rigorous representation of the drag force would increase the accuracy of our simulation, but doing so would markedly increase the complexity of the algorithm and thus make real-time computation more difficult.
- We have ignored the abundances of droplet sizes in considering discomfort. If one droplet would spray passersby, we assume that enough droplets would spray passersby to make them uncomfortable. In fact, it is only significant numbers of droplets that discomfort passersby but we do not know how many droplets would be released nor how many would be needed to be discomforting.

### **References**

Goldstein, S., ed. "Modern Developments in Fluid Dynamics, Volume II." New York: Dover Publications, 1965

Hughes, W. F. and J. A. Brighton. "Fluid Dynamics." New York: McGraw-Hill, 1967.

Lide, D., ed. "CRC." 80<sup>th</sup> Ed. New York: CRC Press, 1999.

Winters, L. "Theory of velocity dependent drag forces."  
<http://courses.ncssm.edu/ph220/labs/vlab1/theory.pdf>. Accessed 2002.

Weather.com Glossary. <http://www.weather.com/glossary/>. Accessed 2002.

## Appendix 1: Relevant Constants, Variables, and Parameters

Physical Constants	Description	Value
$\eta_a$	Viscosity of air	$1.849 \cdot 10^{-5} \text{ kg/m}\cdot\text{s}$ [Lide, 1999]
$\rho_w$	Density of water	$1000 \text{ kg/m}^3$
$\rho_a$	Density of air	$1.2 \cdot 10^{-6} \text{ kg/m}^3$
$g$	Acceleration due to gravity	$9.80 \text{ m/s}$

Situational Constants	Description	Units
$A$	Cross-sectional area of fountain nozzle	$\text{m}^2$
$f_{\max}$	Maximum flow rate of the fountain's pump	$\text{m}^3/\text{s}$
$R_p$	Radius of fountain pool	$\text{m}$
$r$	Radius of smallest uncomfortable water droplet	$\text{m}$
$dt$	Sampling interval of anemometer	$\text{s}$
$k$	$k \equiv \frac{9\eta_a}{2\rho_w r^2}$	$\text{s}^{-1}$

Situational Variables	Description	Units
$v_a$	Instantaneous wind speed	$\text{m/s}$
$f$	Instantaneous flow rate of water from the fountain	$\text{m}^3/\text{s}$
$n$	$n \equiv \frac{g}{k} + \frac{f}{A}$	$\text{m/s}$

Dynamic Variables	Descriptions (respectively)	Units
$x(t), y(t)$	Droplet's horizontal and vertical positions	$\text{m}$
$v_x(t), v_y(t)$	Droplet's horizontal and vertical velocities	$\text{m/s}$
$a_x(t), a_y(t)$	Droplet's horizontal and vertical accelerations	$\text{m/s}^2$

Situational Parameters	Descriptions	Units
$\tau_d$	Default sample wind velocity buffer time	$\text{s}$
$\tau_i$	Buffer time for quickly increasing sample wind velocities	$\text{s}$
$K$	Weight constant	None - dimensionless

## Appendix 2: Flow Algorithm C++ Code

```

#include <iostream>
#include <fstream>
#include <cmath>

double A, Fmax, Rp, r, dt;    //A is the fountain nozzle's area, Fmax is
                             //the //maximum flow rate of the fountain, Rp is
                             //the //radius of the pool, r is the radius of
                             //the //smallest uncomfortable droplet, and dt is
                             //the //sampling rate of the anemometer.

const double eta=.00001849;  //The viscosity of air (kg/m*s)
const double rho=1000;      //The density of water (kg/m^3)
const double g=9.8;        //The acceleration of gravity (m/s^2)

double fountain(double ti, double td, double K, double va, double* vel);
                             //This computes the flow values
void shift(double* vel, double va);
                             //This will shift each value in the
                             //velocity
                             //vector up by one
double choose(double ti, double td, double* vel);
                             //This chooses whether ti or td is used
in our
                             //weighting
double Vstar(double tau, double K, double* vel);
                             //This computes the velocity weighting
and sums
                             //them to find v_star
double weight(double tau, double i, double K);
                             //This finds the weight of a velocity at
a
                             //given time
double flow(double v_star);
                             //Given weighted air velocity v_star,
this
                             //finds what the flow should be
using namespace std;        //introduces namespace std

int main( void )
{
    double velocity[100];    //Creates the velocity buffer,
                             //which is initialized to 100
                             //elements, more than realistically
                             //needed.

    double* vel = velocity;

    ifstream wind, openpar, constants;
    wind.open("wind.txt");  //Contains our windspeed data, va
    openpar.open("parameters.txt"); //Contains tau_i, tau_d, and K, the
                             //buffer parameters
    constants.open("constants.txt"); //Contains the sit'l. constants

    constants >> A >> Fmax >> Rp >> r >> dt;
                             //Reads in values for the
                             //situational constants

    ofstream OutFlow;
    OutFlow.open("flow.txt",ios::out); //Where the flow values are output

    double ti,td,K;

```

```

double va;

wind.clear(); //These two lines reset the cursor to the
wind.seekg(0,ios::beg); //beginning of wind.txt for the next
//iteration.

wind >> va; //Reads in the next wind speed
for(int p=0;p<100;p++)
{
    velocity[p]=va; //This initializes the buffer to va
}
wind.clear(); //Resets the cursor again
wind.seekg(0,ios::beg);

while(wind >> va) //While there are still wind speeds
{ //to read, continue the simulation
    OutFlow << fountain(ti,td,K,va,vel) << endl;
}

wind.close(); //Close all the files
OutFlow.close();
openpar.close();
constants.close();
return 0;
}

//One iteration of the flow algorithm - returns a flow rate given the
given //parameters.
double fountain(double ti, double td, double K, double va, double* vel)
{
    shift(vel,va);
    return flow(Vstar(choose(ti,td,vel),K,vel));
}

//Shifts all the values in the buffer back one line and adds the current wind
//speed to the beginning of the buffer.
void shift(double* vel, double va)
{
    for(int p=98;p>=0;p--)
    {
        vel[p+1]=vel[p];
    }
    vel[0]=va; //Sets the first value in the
} //buffer to the present wind speed

//Checks to see whether the wind is increasing sufficiently and from this
//determines which tau to use.
double choose(double ti, double td, double* vel)
{
    double diff;
    diff = vel[0]-vel[(int) (.4/dt)];
    //Compares the difference in wind
    //speeds over a time of .4 sec
    if((vel[0]>vel[(int) (.2/dt)])&&(vel[(int) (.2/dt)]>vel[(int) (.2/dt)])
    &&(diff>1))
    {
        return ti; //Return tau_i if the wind has
    //increased sufficiently in the
    //last .4 sec
    }
    else
    {

```

```

        return td;                //Otherwise return the default value
    }
}

//Gives the buffered weight value for a given time, i intervals of dt
//after tau.
double weight(double tau, double i, double K)
{
    double n,d;                    //Numerator and denominator
    n=(K+i/(tau-dt)) * dt;
    d=tau*(K+.5);
    return n/d;
}

//Finds v*, the weighted average of the velocities in the buffer.
double Vstar(double tau, double K, double* vel)
{
    double sum=0;
    for(int i=0;i<(int) (tau/dt+1);i++)
        //The number of elements in the
        //buffer is tau divided by the
        //sampling //rate
    {
        sum+=weight(tau,dt*i,K)*vel[i];
    }
    return sum;
}

//Computes the optimal flow rate for a given v*
double flow(double v_star)
{
    double k = 9 * eta / (2 * rho * r * r);
    double radical;
    radical = sqrt(2* v_star *k/Rp);
    double Flow = (A*g/(radical-k));
    if ((k * Rp /2) < v_star)
    {
        return min(Flow, Fmax);    //Cap our flow rate at Fmax
    }
    else
    {
        return Fmax;
    }
}

```

### Appendix 3: C++ Code for the Parameter Testing Algorithm

```

#include <iostream>
#include <fstream>
#include <cmath>

//Physical Constants
const double eta=.00001849; //The viscosity of air (kg/m*s)
const double rho=1000; //the density of water (kg/m^3)
const double g=9.8; //the acceleration of gravity (m/s^2)

const int WindPeriod=2000; //The number of wind speed data points in
//wind.txt

using namespace std; //introduces namespace std
int main( void )
{
    ifstream wind, openpar, constants, inflow;
    wind.open("wind.txt"); //Contains the wind speed data
    openpar.open("parameters.txt"); //Contains ti, td, and K
    constants.open("constants.txt"); //Contains the sit'l. constants

    double A, Fmax, Rp, r, dt;

    //The situational constants are read in from constants.txt
    constants >> A >> Fmax >> Rp >> r >> dt;

    double k = (9/2*eta)/(rho*(pow(r,2))); //Defining k

    inflow.open("flow.txt"); //Contains the flow data

    ofstream badwrite, xItr;
    badwrite.open("Bad.txt"); //Bad.txt holds the inaccuracies
    //resulting from each parameter
    //combination

    //All of the variables we use:
    double n,va,a,v,x,f,Bad,Bmax,Btot,ti,td,K,tw;
    int t;

    //Reads in a new combination of parameters until the end of the file,
    //when it exits the loop.
    while (openpar >> ti >> td >> K)
    {
        Bmax=0; //Keeps track of the worst value
        Btot=0; //Keeps track of the total values

        for(int j=0;j<WindPeriod;j++) //Checks each droplet
        {
            Bad=0;
            wind.clear(); //Resets the cursor
            wind.seekg(0,ios::beg);

            //Increments the cursor from the beginning of wind.txt to
            //the correct position.
            for(int m=0;m<j+1;m++)
            {
                wind >> va;
            }
        }
    }
}

```

```

inflow >> f;           //Reads in the current flow value.

n=g/k+f/A;           //Computes n
tw=2/k*(1-g/(n*k)); //Computes tw

a=k*va;
v=0;
x=0;

//Increments the equations of motion of the particle until
//it hits the ground, at tw.
for(t=1;t<(int) (tw/dt);t++) //Divide by dt because time
{                             //is in units of dt.
    wind >> va;               //Reads in the next va

    v+=a*dt;
    x+=v*dt;
    a=k*(va-v);
}

if (x>Rp)             //If the droplet has overshot
{
    Bad=x-Rp;         //Find how far outside the
                    //pool it landed.

    if(Bad>Bmax)     //If it is worse than the
    {                 //current worst, set it to be
        Bmax=Bad;    //the worst.
    }
    Btot+=Bad;       //Adds the current difference
                    //to the total.
}
}
//Outputs the important values to bad.txt.
badwrite << ti << "\t" << td << "\t" << K << "\t" << Bmax << "\t"
    << Btot << endl;
}
return 0;
}

```