

Two Methods for Determining Impact Time in the Bouncing Ball System

David R. Morrison

May 9, 2008

1 Introduction

Many physical systems can be modelled relatively simply and accurately by a ball bouncing off a surface; hence, the dynamics of this system have been studied by mathematicians and physicists for several decades. If the surface that the ball is rebounding from is fixed, the problem is simple, and can be analyzed as a damped harmonic oscillator. However, if the surface itself is vibrating, very complex behavior can arise. There are two cases to consider: elastic collisions (where no energy is lost due to impact), and inelastic collisions. Both cases have been studied quite extensively, and chaotic behavior has been shown to arise for various parameter values in both circumstances. Holmes (1982) gives a good introduction to the complexities.

The bouncing ball problem has a number of significant applications, from the field of granular flow physics to robotics (through analogies to a person walking). In particular, it is important to be able to model a person walking over uneven or vibrating terrain if robotic technology is to be able to eventually learn to “walk.” Therefore, the bouncing ball on a vibrating platform is an important problem to study and understand, even though much work has been done on the problem already.

In our research, however, we were unable to find a modern simulator that effectively models the behavior of the ball and platform. At least two older programs exist; one accompanies a text called *An Experimental Approach to Nonlinear Dynamics and Chaos*, and is on a 3.5” floppy diskette that only runs on old Apple computers (Tufillaro et al., 1992). A similar program is available at Thomas (2000) that will plot phase diagrams for the bouncing ball program, but the code is poorly-maintained and not as extensive as the original Bouncing Ball software. Therefore, we endeavor to create a

bouncing ball simulator that can be used as a tool in the classroom and in research towards understanding this complex problem.

In the following section, we describe a basic overview of the mathematics behind the model, and in Section 3, we describe our implementation of the simulator. Finally, Section 4 compares two different methods of approximating the system numerically.

2 The Model

The system that we wish to model involves a ball bouncing off a platform vibrating in an oscillatory manner. In order to create a tractable model for this system, we need to make a number of simplifying assumptions: first, we assume that with respect to the mass of the table, the mass of the ball is negligible. This allows us to ignore any effects on the table's velocity caused by the impact. Additionally, we assume that there is no sideways motion of the ball – that is, after each impact, the ball bounces vertically up, and that the table imparts no force in the horizontal direction. We finally assume that the distance the ball travels is large compared to the oscillation of the table; this allows us to approximate the time of the next impact as if the table were motionless (Holmes, 1982).

Under these assumptions, we are therefore able to model the dynamics of the system through the following equation (note that all velocities are measured in the upwards direction):

$$v_b^-(t_j) - v_t(t_j) = -\alpha(v_b^+(t_j) - v_t(t_j)), \quad (1)$$

where t_j is the time of the j^{th} impact, v_b^+ is the velocity of the approaching ball at time t_j , v_b^- is the velocity of the departing ball at this time, and v_t is the velocity of the table at time t_j . We also define $\alpha \in (0, 1]$ to be the **coefficient of restitution** of the system, which determines whether or not the collisions are elastic or inelastic. When $\alpha = 1$, we have perfectly elastic collisions, and $\alpha < 1$ describes an inelastic collision. In equation (1), this simply means that the relative impact velocity (on the right-hand side of the equation) is damped slightly in an inelastic collision. Also note that as long as $v_t > v_b^-$, which must occur if the ball is to hit the table, the equation is consistent (Holmes, 1982).

We can solve equation (1) for v_b^- , and thus determine the outbound velocity in terms of the incoming velocity and the table velocity.

$$v_b^-(t_j) = -\alpha(v_b^+(t_j) - v_t(t_j)) + v_t(t_j) \quad (2)$$

Intuitively, this makes sense: the velocity of the ball’s rebound is simply the sum of the ball’s velocity relative to the table’s frame of reference, plus whatever velocity is imparted to the ball by the motion of the table. Thus, our simulator can numerically determine the table’s velocity and the ball’s velocity at the time of impact, and substitute into equation (2) to determine the rebound velocity v_b^- . The motion of the ball while it is in the air is governed by

$$h_b(t) = g(t - t_j)^2/2 + v_b^+(t_j)(t - t_j) + h_b(t_j), \quad (3)$$

where h_b is the height of the ball at time t , and t_j is the time of the last impact.

Additionally, we have the position of the table as

$$h_t(t) = A \sin(\omega t), \quad (4)$$

where A is the amplitude of oscillation and ω is the angular velocity, defined to be 2π times the the frequency of oscillation.

By equating (3) and (4), we can solve for the next time of impact. Because the system is non-linear, however, this results in chaotic behavior under most circumstances. Additionally, it means that we cannot solve for the time of impact easily; thus, we must employ numerical methods to approximate the next time of impact. This is the subject of the next section. For a more complete discussion of the implications of this system as well as a proof that chaotic behavior arises, see Holmes (1982) or Tuffiaro et al. (1992).

3 The Program

There are a number of tricky issues involved with coding a simulation system that demonstrates complex or chaotic behavior. Most of these issues arise due to imprecision in storing real numbers in a computational system, since it is impossible to store numbers to infinite precision on a computer, due to memory constraints. In general, numbers can be stored to an arbitrary precision, but as more digits are stored, the slower computation becomes.

The traditional method for storing numbers is the **floating-point** representation. For computer systems, this stores numbers in base 2 out to some precision p (p depends on the language and architecture of the computer—see (Sun Microsystems, 2000) for a more complete discussion of floating-point arithmetic). However, this means that any number whose base-2 decimal expansion does not terminate in p digits must be approximated by the

system. In general, this approximation is not a huge impedance to software, but any sort of scientific or mathematics software must consider this issue to avoid strange bugs in the program. It is even more critical to consider these issues when dealing with potentially chaotic behavior, as small perturbations can cause drastic changes in the outcomes of the system.

For our simulator, this has one very important implication: it means that we must be extremely careful how we perform our approximations (especially when determining the time and velocity at an impact). Indeed, when comparing the results of our simulator to the one developed in Thomas (2000), we discovered that even very small changes to the method of approximation produced drastically different results. Additionally, we achieve completely different (yet fascinating) results by using an approximation presented by Holmes instead of the one used by Thomas. This is discussed in Section 4.

First, however, we present an overview of our algorithm. Our code is written in Python, and the algorithm is slightly modified from the code found at Thomas (2000). Below we present pseudo-code for our simulator:

Algorithm 1 BOUNCING BALL SIMULATOR

```

procedure RUN
  time = 0
  dt =  $\frac{1}{2^n}$  ▷ Increment time by this much each step
  while Running do
    time += dt
    Calculate ball position: equation (3)
    Calculate table position: equation (4)
    if ball overlaps table then
      time -= dt ▷ Back up one time-step
      dt /= 2 ▷ Decrease step size for higher precision
      if Close enough then ▷ Collision
        Calculate velocity after impact: equation (2)
      end if
    end if
  end while
end procedure

```

The basic operation of the code is fairly simple. We keep a “time” counter that increases in small, discrete increments. At each step, we update the position of the ball and platform on the screen, and check for a collision between the two. Since we are operating in base 2, having a time-step size of the form $\frac{1}{2^n}$ ensures that the decimal does not have to be approximated,

and thus avoids loss of precision.

Unfortunately, in order to refresh the screen at a reasonable rate, this step size must be fairly large; thus, when we detect a collision with the platform, there is a large range of values that could be the *actual* position of the ball at the collision. This leads to a significant loss of precision when updating the phase portraits at each collision, rendering the image at very low resolution. This is not overly useful. Thus, in the next section we discuss two possible ways of approximating this collision time.

4 Collision-time approximations

The approach taken in Thomas (2000) to approximate the time of collision is the same as the approach presented above: if a collision occurs, back up one step and narrow the time resolution. Repeat this process until we are within ϵ of the actual value. This is a process we call **time-resolution shrinking** and it ensures that we are able to get a reasonable estimate of the collision time.

The problem with this approach is that it has the potential to be very time-consuming; after some experimentation, it became apparent that the computation expended in performing this resolution-narrowing takes about a half-second to complete, resulting in a small jerk at every impact. However, by plotting a phase portrait of t_j versus t_{j-1} , we can gain a picture of the behavior of the system over time. An example of this is shown in Figure 1; this is a picture of a strange attractor that arises from the system (see Thomas (2000)).

However, the time-resolution shrink method still seems like an incredibly inelegant solution. In Holmes (1982), it is stated that (under the assumptions presented in Section 2), we can approximate the time of the next impact using a single equation, which is extremely fast (it takes on the order of 10^{-9} seconds to perform this calculation):

$$t_{j+1} = \frac{2v_b^-}{g} + t_j \quad (5)$$

Note that this assumption is the limiting case as $\omega \rightarrow 0$. In other words, if the ball is bouncing significantly higher than the platform, we can approximate the next time of impact as though the platform were at a fixed height.

In Figure 2, we show the plot produced by this method using the same parameters that generated Figure 2. Clearly this is an extremely different

qualitative result! The difference becomes even more stark when we notice that the plot is on a completely different scale and shift from the first.

Clearly, the simulation is extremely sensitive to the choice of approximation. Unfortunately, we have no idea why these two approximation methods produce such drastically different, yet beautiful results. Our initial hypothesis based on observations is that there are a number of cases when the ball is not moving significantly with respect to the platform, thus invalidating this approximation. This is further supported by the fact that there are a number of negative times that are shown in Figure 2, but this must be studied more before anything definite can be said, and is a subject of future work for the project.

There are two approaches that could be taken to check this hypothesis; first, compare the results of impact times and velocities to determine where they diverge, and how this divergence occurs. Secondly, we need to experiment with a much wider range of parameter values to determine how these systems behave under different conditions. If we could find a system in which the large-bounce approximation holds for a long period of time, the results from the two approximation methods could be much better-compared.

5 Conclusion

The bouncing ball and vibrating platform is an extremely important model in dynamical systems, as well as in many other physical sciences; thus, it is important for mathematicians and scientists who study this system to have a good simulator of the dynamics observed for the system.

We developed a simulator in Python that both shows the action of the system, as well as plots a number of different phase portraits to demonstrate the chaotic behavior of the system for various parameter values. Furthermore, we have implemented a new approximation for impact time and compared it to a resolution-shrinking method described in Thomas (2000). However, due to computer precision and some (potentially) invalid assumptions, this appears to produce a system that is non-physical, and must be improved.

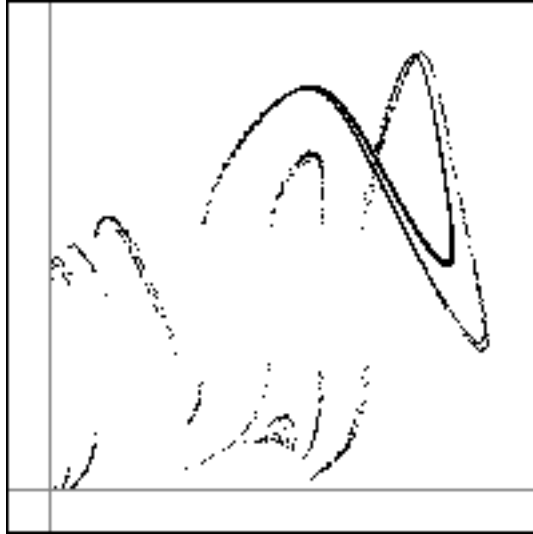


Figure 1: A plot of t_j versus t_{j-1} after 50,000 iterations, using parameter values $A = 0.012$ and $\omega = 2\pi \cdot 6.03987$. This plot was developed using the time-resolution shrinkage method of approximation.

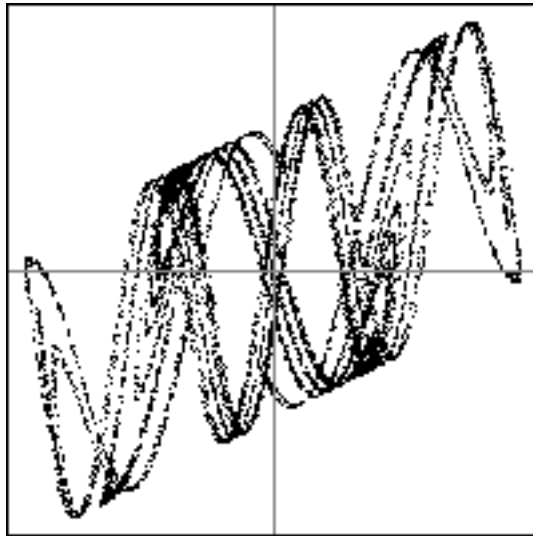


Figure 2: A plot of t_j versus t_{j-1} after 10,000 iterations, using parameter values $A = 0.012$ and $\omega = 2\pi \cdot 6.03987$. This plot was developed using the Holmes approximation in equation 5.

References

- Holmes, P.J. 1982. The dynamics of repeated impacts with a sinusoidally vibrating able. *Journal of Sound and Vibration* 84(2):173–189.
- Sun Microsystems. 2000. Numerical computation guide. <http://docs.sun.com/source/806-3568/ncgT0C.html>. Last Accessed: 25 April, 2008.
- Thomas, Joseph. 2000. Bouncing ball. http://chaos.phy.ohiou.edu/~thomas/chaos/bouncing_ball.html. Last Accessed: 24 April, 2008.
- Tufillaro, N. B., T. Abbott, and J. P. Reilly. 1992. *An Experimental Approach to Nonlinear Dynamics and Chaos*. Addison-Wesley, Redwood.