Team # 210
Problem B
Trebuchet: The Dynamics of a Medieval Siege Engine

Abstract

Our task for this problem was to design a trebuchet that does maximum damage, given that its counterweight has a mass of 5,000 kilograms and is released from a height of 2 meters above the ground. To that end, we first modeled the trebuchet as a uniform, frictionless beam, pinned at its fulcrum, with the counterweight (taken to be a point mass) on one end of the beam, and a sling of fixed length and negligible mass attached to the other end. The projectile (located on the far end of the sling) was also taken to be a point mass. We assumed that the beam could make a full revolution about the fulcrum with neither the counterweight nor the projectile touching the ground, and that the beam would not fail during operation. Moreover, we interpreted maximizing the damage done by the projectile as maximizing its speed upon release, which we assumed to occur when the counterweight reached its lowest point for the first time.

Lagrangian mechanics was used to find the governing differential equations for this system, and an algorithm for solving said differential equations using first-order Taylor series approximations was developed and translated into C++ code. Finally, the speed of the projectile upon release was calculated for all allowed dimensions of the trebuchet (for a chosen set of parameters), and the dimensions corresponding to the maximum observed speed were recorded. For a steel beam with square cross sections of side length 10 centimeters whose fulcrum was located 1.5 meters above the ground, and a 100-kilogram projectile, we found that the optimum lengths of the arm of the counterweight, the opposing arm, and the sling, were approximately 0.51 meters, 0.91 meters, and 0.58 meters, respectively. We conclude that, for the parameters we chose, the ratio of the length of the sling to that of the arm to which the sling is attached should

be about 2:3, that together these should be as long as possible, and that the lever arm of the counterweight should be shorter than the opposing arm. To find out whether this works as a general rule-of-thumb would require further investigation.

The strengths of our approach include that it takes into account the beam's size and shape. It is also quite general, in the sense that it makes very few simplifications beyond the idealized model of the system and the approximate numerical methods used to solve for the motion of the system. The weaknesses of this approach are that it fails to account for friction between the beam and the fulcrum, the mass of the sling, and the sling's elasticity. In addition, we have made no attempt here to determine whether or not the trebuchet we have designed will withstand operation conditions. Nevertheless, as a first approximation, our approach appears to do very well. An actual trebuchet would need to be built to check the predictions of our model.

Problem Definition

Consider the trebuchet represented in Figure 1. It consists of a beam of mass m_2 which is pinned at the point O (the origin of the xyz-coordinate system, and henceforth referred to as the fulcrum) at some distance h_0 above the ground. On one side of the beam, at a distance r_1 from the fulcrum, is a counterweight of mass m_1. Attached to the other side of the beam, at a distance r_2 from the fulcrum, is a sling of length r_3. Inside the sling is a projectile of mass m_3. The counterweight is to be released (presumably from rest) from a height h above the ground, launching the projectile into the air. We would like to find the lengths r_1, r_2, and r_3 which enable the projectile to do the most possible damage, given that $m_1 = 5{,}000\,\text{kg}$ and $h = 2\,\text{m}$.

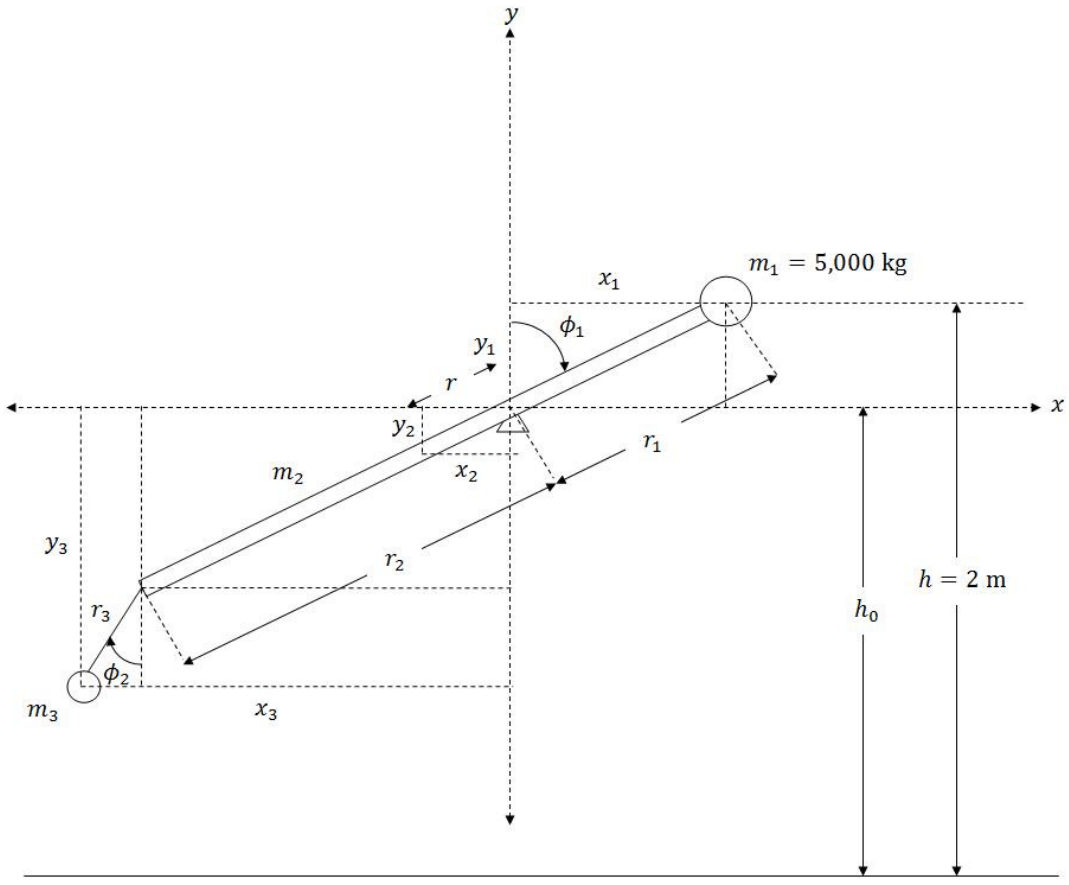


Figure 1. Diagram of simplified trebuchet, with parameters labeled.

To simplify our analysis, we will treat the counterweight and the projectile as point masses, and the beam as a frictionless, slender rod whose mass is evenly distributed along its length. We will also neglect the mass of the sling, and we'll assume that its length remains constant. Furthermore, we will suppose that the beam can make a full revolution about the fulcrum with neither the counterweight nor the projectile touching the ground. Finally, we will assume that the beam will not fail during operation.

Now in order to do the most damage, the projectile must have as much energy as possible. Since we are treating the projectile as a point mass (thereby ignoring any rotation it may acquire about its center of mass), we must maximize its translational speed at the point of release. To be definite, we will take the point of release to be the point at which the counterweight reaches its lowest height above the ground for the first time. In order to find numerical values for r_1, r_2, and r_3, we will eventually need to decide on a material for the beam and the shape of its cross section, as well as values for h_0 and m_3. For now, however, we will treat all parameters as unknowns, both for mathematical convenience and so that the results of our analysis will be general. We are now ready to proceed to an analysis of the trebuchet system.

Analysis

Since the only external force on the trebuchet system (i.e., gravity) is conservative, it is only natural to use the Lagrangian formulation of mechanics to analyze it. We choose as our generalized coordinates ϕ_1, the angle the arm of length r_1 makes with the positive y-axis, and ϕ_2, the angle the sling makes with the negative y-axis, both measured clockwise (see Figure 1).

We begin by expressing the Cartesian coordinates of the counterweight, the center of mass of the beam, and the projectile (which is located at the end of the sling) in terms of the

generalized coordinates ϕ_1 and ϕ_2. The position of the counterweight is given by (x_1, y_1), where

$$x_1 = r_1 \sin\phi_1 \tag{1}$$

$$y_1 = r_1 \cos\phi_1 \tag{2}$$

We will also need the first time derivatives of these coordinates, so we might as well calculate them right now:

$$\dot{x}_1 = r_1\dot{\phi}_1 \cos\phi_1 \tag{3}$$

$$\dot{y}_1 = -r_1\dot{\phi}_1 \sin\phi_1 \tag{4}$$

Now the center of mass of the beam will be a distance r from the fulcrum in the direction of the sling, where

$$r = \frac{r_2 - r_1}{2} \tag{5}$$

Thus, the position of the center of mass of the beam is given by (x_2, y_2), where

$$x_2 = -r \sin\phi_1 \tag{6}$$

$$y_2 = -r \cos\phi_1 \tag{7}$$

Taking the first time derivatives of (6) and (7), we find that

$$\dot{x}_2 = -r\dot{\phi}_1 \cos\phi_1 \tag{8}$$

$$\dot{y}_2 = r\dot{\phi}_1 \sin\phi_1 \tag{9}$$

Finally, the position of the projectile (at the end of the sling) is given by (x_3, y_3), where

$$x_3 = -(r_2 \sin\phi_1 + r_3 \sin\phi_2) \tag{10}$$

$$y_3 = -(r_2 \cos\phi_1 + r_3 \cos\phi_2) \tag{11}$$

Again, taking the first time derivatives of (10) and (11), we find that

$$\dot{x}_3 = -(r_2\dot{\phi}_1 \cos\phi_1 + r_3\dot{\phi}_2 \cos\phi_2) \tag{12}$$

$$\dot{y}_3 = r_2\dot{\phi}_1 \sin\phi_1 + r_3\dot{\phi}_2 \sin\phi_2 \tag{13}$$

We must now calculate the total kinetic and potential energies of the system. We begin with kinetic energy of the counterweight, which is purely translational because the counterweight is treated as a point mass:

$$T_1 = \frac{1}{2} m_1 v_1^{\,2} = \frac{1}{2} m_1 \left(\dot{x}_1^{\,2} + \dot{y}_1^{\,2} \right) = \frac{1}{2} m_1 r_1^{\,2} \dot{\phi}_1^{\,2} \tag{14}$$

Now the beam is not a point mass, but rather a rigid body which is rotating about the z-axis as shown in Figure 1. Its kinetic energy is therefore purely rotational, and is given by

$$T_2 = \frac{1}{2} I_{zz} \dot{\phi}_1^{\,2} \tag{15}$$

where I_{zz} is the moment of inertia of the beam about the z-axis. The moment of inertia of a slender rod of mass m and length ℓ about an axis perpendicular to its length and passing through its center of mass is given by

$$I'_{zz} = \frac{1}{12} m\ell^2 \tag{16}$$

(Resnick, 185). By the Parallel-Axis Theorem, its moment of inertia about a parallel axis a distance r from its center of mass is given by

$$I_{zz} = I'_{zz} + mr^2 = \frac{1}{12} m\ell^2 + mr^2 \tag{17}$$

Substituting m_2 for m, $(r_1 + r_2)$ for ℓ, and r for r, we have the moment of inertia of our beam about the z-axis:

$$I_{zz} = \frac{1}{12} m_2 (r_1 + r_2)^2 + m_2 r^2 \tag{18}$$

Even though we have derived this equation for the beam's moment of inertia, we will leave it as I_{zz} for now so that our results will generalize to other kinds of beams.

The kinetic energy of the point mass projectile (while it is still attached to the sling) is purely translational, and is therefore given by the following:

$$T_3 = \frac{1}{2} m_3 v_3^{\,2} = \frac{1}{2} m_3 \left(\dot{x}_3^{\,2} + \dot{y}_3^{\,2} \right) \tag{19}$$

After substituting (12) and (13) and simplifying, this becomes

$$T_3 = \frac{1}{2} m_3 \left[r_2^{\,2} \dot{\phi}_1^{\,2} + 2 r_2 r_3 \dot{\phi}_1 \dot{\phi}_2 \cos(\phi_1 - \phi_2) + r_3^{\,2} \dot{\phi}_2^{\,2} \right] \tag{20}$$

Here we have used the trigonometric identity that

$$\cos\phi_1 \cos\phi_2 + \sin\phi_1 \sin\phi_2 = \cos(\phi_1 - \phi_2) \tag{21}$$

(Taylor, front cover). The total kinetic energy of the system is simply the sum of the kinetic energies of the counterweight, beam, and projectile:

$$T = T_1 + T_2 + T_3 \tag{22}$$

Now we turn our attention to (gravitational) potential energy, which we will measure with respect to the line $y = 0$ rather than ground level for convenience. The potential energies of the counterweight, the center of mass of the beam, and the projectile are, respectively,

$$U_1 = m_1 g y_1 = m_1 g r_1 \cos\phi_1 \tag{23}$$

$$U_2 = m_2 g y_2 = -m_2 g r \cos\phi_1 \tag{24}$$

$$U_3 = m_3 g y_3 = -m_3 g \left(r_2 \cos\phi_1 + r_3 \cos\phi_2 \right) \tag{25}$$

where we have used the results of (2), (7), and (11). The total potential energy of the system, then, is the sum of these:

$$U = U_1 + U_2 + U_3 \tag{26}$$

We now have everything we need to write down the Lagrangian for the system:

$$L = T - U \tag{27}$$

Substituting (22) and (26)—and by extension, (14), (15), (20), (23), (24), and (25)—we have

$$\begin{aligned} L = {} & \frac{1}{2} m_1 r_1^{\,2} \dot{\phi}_1^{\,2} + \frac{1}{2} I_{zz} \dot{\phi}_1^{\,2} + \frac{1}{2} m_3 \left[r_2^{\,2} \dot{\phi}_1^{\,2} + 2 r_2 r_3 \dot{\phi}_1 \dot{\phi}_2 \cos(\phi_1 - \phi_2) + r_3^{\,2} \dot{\phi}_2^{\,2} \right] \\ & - m_1 g r_1 \cos\phi_1 + m_2 g r \cos\phi_1 + m_3 g (r_2 \cos\phi_1 + r_3 \cos\phi_2) \end{aligned} \tag{28}$$

The two Lagrangian equations, which determine the generalized coordinates as functions of time, are as follows:

$$\frac{\partial L}{\partial \phi_1} = \frac{d}{dt} \frac{\partial L}{\partial \dot{\phi}_1} \tag{29}$$

$$\frac{\partial L}{\partial \phi_2} = \frac{d}{dt} \frac{\partial L}{\partial \dot{\phi}_2} \tag{30}$$

Focusing first on ϕ_1, we find that

$$\frac{\partial L}{\partial \phi_1} = -m_3 r_2 r_3 \dot{\phi}_1 \dot{\phi}_2 \sin(\phi_1 - \phi_2) + (m_1 r_1 - m_2 r - m_3 r_2) g \sin\phi_1 \tag{31}$$

and

$$\frac{\partial L}{\partial \dot{\phi}_1} = \left(m_1 r_1^{\,2} + I_{zz} + m_3 r_2^{\,2} \right) \dot{\phi}_1 + m_3 r_2 r_3 \dot{\phi}_2 \cos(\phi_1 - \phi_2) \tag{32}$$

Taking the first time derivative of (32), and substituting the result along with (31) into (29), we obtain, after some simplification, the following:

$$(m_1 r_1 - m_2 r - m_3 r_2) g \sin\phi_1 = \left(m_1 r_1^{\,2} + I_{zz} + m_3 r_2^{\,2} \right) \ddot{\phi}_1 + m_3 r_2 r_3 \left[\ddot{\phi}_2 \cos(\phi_1 - \phi_2) + \dot{\phi}_2^{\,2} \sin(\phi_1 - \phi_2) \right] \tag{33}$$

Similarly, for ϕ_2, we find that

$$\frac{\partial L}{\partial \phi_2} = m_3 r_2 r_3 \dot{\phi}_1 \dot{\phi}_2 \sin(\phi_1 - \phi_2) - m_3 g r_3 \sin\phi_2 \tag{34}$$

and

$$\frac{\partial L}{\partial \dot{\phi}_2} = m_3 r_2 r_3 \dot{\phi}_1 \cos(\phi_1 - \phi_2) + m_3 r_3^{\,2} \dot{\phi}_2 \,) \tag{35}$$

Taking the time derivative of (35), substituting the result and (34) into (30), and simplifying, we find that

$$-g \sin \phi_2 = r_3 \ddot{\phi}_2 + r_2 \left[\ddot{\phi}_1 \cos(\phi_1 - \phi_2) - \dot{\phi}_1^{\,2} \sin(\phi_1 - \phi_2) \right] \tag{36}$$

Now (33) and (36) constitute a set of two coupled, nonlinear, second-order differential equations for ϕ_1 and ϕ_2. If we could solve them analytically for $\phi_1(t)$ and $\phi_2(t)$, we could find the time at which $\phi_1 = \pi$ (the point at which we have supposed that the projectile is released), write down the speed of the projectile at that time, and proceed to maximize it with respect to r_1, r_2, and r_3. Unfortunately, we cannot solve (33) and (36) analytically; however, we can solve them numerically.

Numerical Analysis

Given m_1, m_2, m_3, r_1, r_2, r_3, $\phi_1(0)$, $\phi_2(0)$, $\dot{\phi}_1(0)$, and $\dot{\phi}_2(0)$, we can solve (33) and (36) numerically using a series of Taylor series approximations. We have already been given that $m_1 = 5{,}000$ kg. Now if we assume that the beam is to be cut from material with uniform cross-section of area A, the value of m_2 (the mass of the beam) will depend on the mass density ρ of the material, the cross-sectional area A, and the lengths r_1 and r_2. In particular,

$$m_2 = \rho A (r_1 + r_2) \tag{37}$$

To obtain a numerical value for m_2, we are forced to decide on a material for the beam and a shape for its cross section. Let us suppose, therefore, that the beam is made of steel, whose mass

density is $\rho = 7{,}850$ kg/m^3 (Gere, 991). Furthermore, we will take the cross section to be a square of side length $s = 0.10$ m (we will see shortly that this is consistent with our approximating the beam as a slender rod). Hence, the cross-section area is given by

$$A = s^2 = (0.10\,\text{m})^2 \tag{38}$$

We must now decide what we would like our projectile to be. For simplicity, let's make it a spherical rock of mass $m_3 = 100$ kg. Now $\phi_1(0)$ will depend on the height h_0 of the fulcrum above the ground and the height h from which the counterweight is released. In particular (refer to Figure 1),

$$\phi_1(0) = \arccos\left(\frac{h - h_0}{r_1}\right) \tag{39}$$

We have been given that $h = 2$ m, so we just need to decide on a value for h_0. Let's say that $h_0 = 1.5$ m. We are free to choose $\phi_2(0)$, so for convenience we will take it to be zero. Likewise, since we assume that the system is released from rest, $\dot{\phi}_1(0) = \dot{\phi}_2(0) = 0$. Next we must obtain from (33) and (36) expressions for $\ddot{\phi}_1(0)$ and $\ddot{\phi}_2(0)$ in terms of everything else. After a lot of simplification, involving the well-known trigonometric identity $1 - \cos^2(\phi_1 - \phi_2) = \sin^2(\phi_1 - \phi_2)$, we find that

$$\ddot{\phi}_1 = \frac{g \sin\phi_1 (m_1 r_1 - m_2 r - m_3 r_2) + m_3 r_2 \left[g \sin\phi_2 \cos(\phi_1 - \phi_2) - r_3 \dot{\phi}_2^{\,2} \sin(\phi_1 - \phi_2) - r_2 \dot{\phi}_1^{\,2} \cos(\phi_1 - \phi_2)\sin(\phi_1 - \phi_2)\right]}{m_1 r_1^{\,2} + I_{zz} + m_3 r_2^{\,2} \sin^2(\phi_1 - \phi_2)} \tag{40}$$

and

$$\ddot{\phi}_2 = \frac{r_2}{r_3}\left[\dot{\phi}_1^{\,2} \sin(\phi_1 - \phi_2) - \ddot{\phi}_1 \cos(\phi_1 - \phi_2)\right] - g \sin\phi_2 \tag{41}$$

where we already know what $\ddot{\phi}_1$ is from (40).

Now we are ready to begin the iterative approximation process. For every possible value of r_1, r_2, and r_3 (we'll discuss what this means shortly), we start by using (40) and (41) to find $\ddot{\phi}_1(0)$ and $\ddot{\phi}_2(0)$ (the initial values of $\ddot{\phi}_1$ and $\ddot{\phi}_2$) based on $\phi_1(0)$, $\phi_2(0)$, $\dot{\phi}_1(0)$, and $\dot{\phi}_2(0)$ (the initial values of ϕ_1, ϕ_2, $\dot{\phi}_1$, and $\dot{\phi}_2$ given above). We then approximate the next values of ϕ_1, ϕ_2, $\dot{\phi}_1$, and $\dot{\phi}_2$ using first-order Taylor series as follows:

$$\phi_1(t_0 + dt) \approx \phi_1(t_0) + \dot{\phi}_1(t_0)dt \tag{42}$$

$$\phi_2(t_0 + dt) \approx \phi_2(t_0) + \dot{\phi}_2(t_0)dt \tag{43}$$

$$\dot{\phi}_1(t_0 + dt) \approx \dot{\phi}_1(t_0) + \ddot{\phi}_1(t_0)dt \tag{44}$$

$$\dot{\phi}_2(t_0 + dt) \approx \dot{\phi}_2(t_0) + \ddot{\phi}_2(t_0)dt \tag{45}$$

Here dt is a constant time increment which we can choose; let's set it at 0.01 s. We then calculate $\ddot{\phi}_1$ and $\ddot{\phi}_2$ again using (40) and (41) and repeat the process until ϕ_1 reaches π to within some tolerance (0.05 radians, say). At that point, we stop and calculate the speed of the projectile according to the following equation (refer to (19) and (20)):

$$v_3 = \sqrt{r_2^2\dot{\phi}_1^2 + 2r_2r_3\dot{\phi}_1\dot{\phi}_2\cos(\phi_1 - \phi_2) + r_3^2\dot{\phi}_2^2} \tag{46}$$

We then use whichever values of r_1, r_2, and r_3 result in the highest value of v_3.

We must now put bounds on r_1, r_2, and r_3. From Figure 1, we see that the smallest possible value of r_1 for a given h_0 is $h - h_0$, and the largest value of r_1 for which the counterweight clears the ground is h_0. We therefore have the following bounds on r_1:

$$h - h_0 < r_1 < h_0 \tag{47}$$

The bounds for r_2 and r_3 are only slightly more complicated. By the same reasoning as for r_1,

the sum $r_2 + r_3$ cannot exceed h_0. If we allow for very small values of r_2, and suppose that the shortest sling length available is ℓ_0 (a reasonable value might be 0.30 m), we have that

$$\ell_0 < r_3 < h_0 \tag{48}$$

and

$$0 < r_2 < h_0 - r_3 \tag{49}$$

Of course, we can't check all of the infinite values contained in (47), (48), and (49), so we'll have to choose a length increment of dr. Since we presumably don't need to be accurate to a millimeter, we can let $dr = 0.01\,\text{m}$, or one centimeter.

One question remains unanswered, namely whether or not we can get accurate results by treating the beam as a slender rod. For the parameters we have chosen, the shortest possible beam would occur when $r_1 = h - h_0 = 0.50\,\text{m}$ and $r_2 = 0\,\text{m}$. The total length of the beam would then be 50 centimeters. The side length on the cross section is 10 centimeters. Thus, the ratio of the two is 5. As a first approximation, this is good enough. If additional accuracy is needed, we can always calculate the moment of inertia of the beam exactly and use that value everywhere the moment of inertia occurs (this is why we left everything in terms of the moment of inertia to begin with).

To summarize, we have created an algorithm for finding the values of r_1, r_2, and r_3 which maximize the speed of the projectile at its release. We just have to translate it into a computer program and let a computer do the rest. Appendix A contains such a program written in the C++ programming language called "optimize_trebuchet.cpp." Note that it was written for the parameter values chosen above, but in such a way that the parameters can be varied with ease. Thus, it can be used to optimize the projectile's speed for any desired set of parameters. The results obtained with the chosen parameter values are discussed in what follows.

Results

For the values of the parameters discussed in the previous section, the optimum lengths were found to be as follows:

$$r_1 \approx 0.51 \text{ m} \tag{50}$$

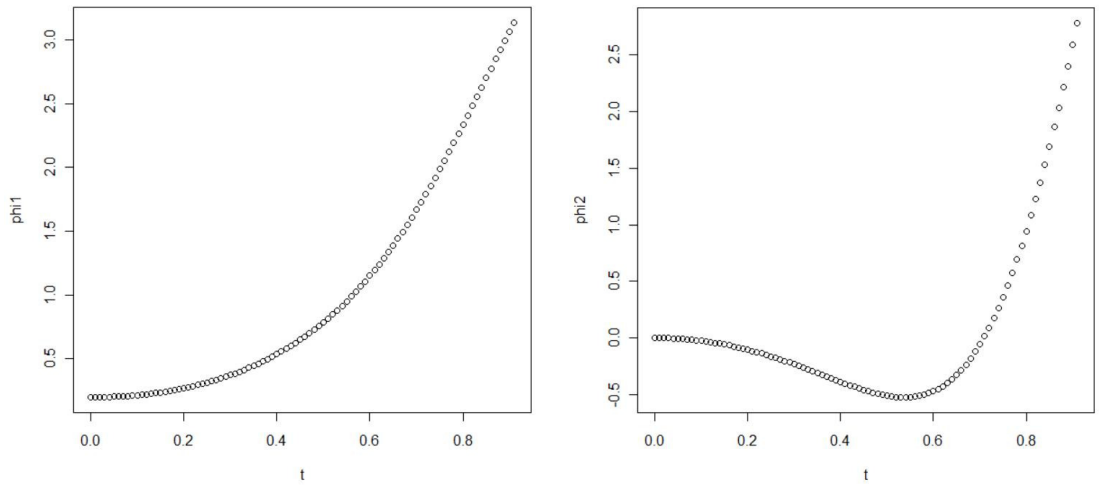
$$r_2 \approx 0.91 \text{ m} \tag{51}$$

$$r_3 \approx 0.58 \text{ m} \tag{52}$$

The corresponding speed of the projectile upon release was

$$v_3 \approx 17.48 \text{ m/s} \tag{53}$$

See Figure 2 for corresponding plots of $\phi_1(t)$ and $\phi_2(t)$. (These plots were generated using the R programming language with the data output by "graph_data.cpp." The code for "graph_data.cpp" can be found in Appendix B, the data itself can be found in Appendix C, and the script used to plot the data in R can be found in Appendix D.)



(a) (b)

Figure 2. (a) Plot of ϕ_1 (radians) versus t (seconds). (b) Plot of ϕ_2 (radians) versus t (seconds).

Obviously, (50), (51), (52), and (53) are just approximations; our model for the trebuchet was an idealization from the start, and the method we used to calculate its motion as a function of time was only approximate. Having said that, the speed given in (53) seems reasonable enough; it is neither astronomically small nor astronomically large. The same is true of the three lengths. We can see that r_1 came out as close to the minimum value as our length increment allowed it to be. Likewise, the total length $r_2 + r_3$ of the sling and the opposing arm was as close as possible to the maximum allowable length, with r_3 approximately two-thirds of r_2. This seems to indicate that for optimal results, r_2 and r_3 should together be as long as possible, with the ratio of r_3 to r_2 about 2:3. In addition, it appears that r_1 should be smaller than r_2. However, we should not be quick to make a generalization based on one simulation. Ideally, we would run multiple simulations with different values for the parameters, and we would check the results of said simulations by actually building the corresponding trebuchets and testing them under realistic conditions. At that point we might be able to arrive at a general rule-of-thumb for how to design the "best" trebuchet. Until then, we can get a rough idea by running the program in Appendix A with any given parameters.

Conclusion

Our task was to design a trebuchet which dealt the most damage when a 5,000-kilogram counterweight was released from a height of 2 meters above the ground. After modeling the trebuchet and creating a recursive algorithm for approximating its motion as a function of time, we found (for the parameters we chose) that the ratio of the length of the sling to that of the arm to which the sling is attached should be about 2:3, that together these should be as long as possible, and that the lever arm of the counterweight should be less than the opposing arm. We

cannot conclude that this result would work as a rule-of-thumb; however, for any given set of parameters, we can use the program in Appendix A to obtain a first approximation to the best trebuchet possible.

Appendix A: Code for "optimize_trebuchet.cpp"

```
  /* optimize_trebuchet.cpp
    This program was written by Team 210 to help solve Problem B of
the 2010 University Physics Competition.  */

  // Declare libraries.
  #include<cmath>
  #include<cstdlib>
  #include<iostream>
  #include<fstream>
  #include<string>

  // Declare the namespace.
  using namespace std;

  // The main program.
  int main()
  {
      // Define physical constants.
      const double PI = acos(-1);   // Pi.
      const double g = 9.8; // The acceleration due to gravity, m/s/s.

      // Declare/define the masses of the system.
      double m1 = 5000;   // The mass of the counterweight in kg.
      double m2;          // The mass of the beam in kg.
      double m3 = 100;    // The mass of the projectile in kg.

      // Define the heights of the system.
      double h = 2;   // The initial height of the counterweight in m.
      double h0 = 1.5;    // The height of the fulcrum in m.

      // Declare the lengths of the system.
      double r1;          // The length of the "short" arm in m.
      double r2;          // The length of the "long" arm in m.
      double r3;          // The length of the sling in m.

      // Declare the parameters that depend on the lengths.
      double r;           // = (r2-r1)/2;
      double Izz;         // = m2*pow((r1+r2),2)/12+m2*pow(r,2);

      // Define tolerance for phi1 (see below).
      double tol = 0.05;

      // Define the maximum time and the time increment.
      double tmax = 10;         // The maximum time in s.
      double dt = tmax/1000;    // The time increment in s.
      double T = 0;

      // Define the dimension of future vectors.
      int dim = (int)(tmax/dt+1);
```

```
      int n;

      // Declare and define the time vector.
      double t[dim];
      t[0] = 0;
      for(int k=1; k<dim; k++)
      {
         t[k]=t[k-1]+dt;
      }

      // Declare the generalized coordinates and time derivatives.
      double phi1[dim];    // phi1
      double phi2[dim];    // phi2
      double phi1d[dim];   // first time derivative of phi1
      double phi2d[dim];   // first time derivative of phi2
      double phi1dd[dim];  // second time derivative of phi1
      double phi2dd[dim];  // second time derivative of phi2

      /* For all possible values of r1, r2, and r3, compute maximum
speed of m3 when phi1 = PI. */

      // Declare variables that will keep track of maximum speed.
      double v3=0;             // The current speed.
      double v3max=0;          // The maximum speed observed.
      double r1max=0;        // The value of r1 corresponding to vmax.
      double r2max=0;        // The value of r2 corresponding to vmax.
      double r3max=0;        // The value of r3 corresponding to vmax.

      // Define the length increment.
      double dr = 0.01;   // The length increment in m.

      for(r1=h-h0+dr;r1<h0;r1+=dr)   // h-h0 < r1 < h0
      {
         for(r3=0.30;r3<h0;r3+=dr)   // 0.30 < r3 < h0
         {
            for(r2=dr;r2<h0-r3;r2+=dr)   // 0 < r2 < h0-r3
            {
               // Determine the mass of the beam.
               m2 = (7850)*pow(0.10,2)*(r1+r2);

               // Determine the parameters that depend on the lengths.
               r = (r2-r1)/2;
               Izz = m2*pow((r1+r2),2)/12+m2*pow(r,2); // Slender rod.

               // Determine the initial value of phi1.
               phi1[0] = acos((h-h0)/r1);

               // Define the initial value of phi2.
               phi2[0] = 0;

               // Define the initial values of phi1d and phi2d.
               phi1d[0] = 0;
```

```
                phi2d[0] = 0;

                // Determine the initial values of phi1dd and phi2dd.
                phi1dd[0] = ( g*sin(phi1[0])*(m1*r1-m2*r-m3*r2) +
m3*r2*( g*sin(phi2[0])*cos(phi1[0]-phi2[0]) -
r3*pow(phi2d[0],2)*sin(phi1[0]-phi2[0]) -
r2*pow(phi1d[0],2)*cos(phi1[0]-phi2[0])*sin(phi1[0]-phi2[0]) ) ) / (
m1*pow(r1,2) + m3*pow(r2,2)*pow(sin(phi1[0]-phi2[0]),2) + Izz );

                phi2dd[0] = ( r2*( pow(phi1d[0],2)*sin(phi1[0]-phi2[0])
- phi1dd[0]*cos(phi1[0]-phi2[0]) ) - g*sin(phi2[0]) ) / r3;

                n = 0;

                // Find subsequent values of coordinates/derivatives.
                for(int k=1; k<dim; k++)
                {
                   // Approximate with first-order Taylor Series.
                   phi1[k] = phi1[k-1] + phi1d[k-1]*dt;
                   phi2[k] = phi2[k-1] + phi2d[k-1]*dt;
                   phi1d[k] = phi1d[k-1] + phi1dd[k-1]*dt;
                   phi2d[k] = phi2d[k-1] + phi2dd[k-1]*dt;

                   // If phi1 has reached PI within tolerance, break.
                   if(abs(PI-phi1[k])<tol)
                   {
                       n = k;  // Store index for future use.
                       break;
                   }

                   // Calculate phi1dd and phi2dd based on above.
                   phi1dd[k] = ( g*sin(phi1[k])*(m1*r1-m2*r-m3*r2) +
m3*r2*( g*sin(phi2[k])*cos(phi1[k]-phi2[k]) -
r3*pow(phi2d[k],2)*sin(phi1[k]-phi2[k]) -
r2*pow(phi1d[k],2)*cos(phi1[k]-phi2[k])*sin(phi1[k]-phi2[k]) ) ) / (
m1*pow(r1,2) + m3*pow(r2,2)*pow(sin(phi1[k]-phi2[k]),2) + Izz );

                   phi2dd[k] = ( r2*( pow(phi1d[k],2)*sin(phi1[k]-
phi2[k]) - phi1dd[k]*cos(phi1[k]-phi2[k]) ) - g*sin(phi2[k]) ) / r3;
                }

                // Calculate the current speed of m3.
                v3 = sqrt( pow(r2*phi1d[n],2) + pow(r3*phi2d[n],2) +
2*r2*r3*phi1d[n]*phi2d[n]*cos(phi1[n]-phi2[n]) );

                // If this speed is greater than the maximum speed
observed thus far, replace the maximum speed observed with this speed.
                if(v3>v3max)
                {
                   v3max = v3;
                   r1max = r1;
                   r2max = r2;
```

```
                    r3max = r3;
                    T = t[n];
                }
            }
        }
    }

    // Display the maximum speed and the corresponding r1, r2, r3.
    cout << endl << "  vmax = " << v3max << endl << "  r1 = " <<
r1max << endl << "  r2 = " << r2max << endl << "  r3 = " << r3max <<
endl << "  tmax = " << T;

    cin.get();
  }
```

Appendix B: Code for "graph_data.cpp"

```
  /* graph_data.cpp

  This program was written by Team 210 to help solve Problem B of the
2010 University Physics Competition.  */

  // Declare libraries.
  #include<cmath>
  #include<cstdlib>
  #include<iostream>
  #include<fstream>
  #include<string>

  // Declare the namespace.
  using namespace std;

  // The main program.
  int main()
  {
      // Define physical constants.
      const double PI = acos(-1);   // Pi.
      const double g = 9.8; // The acceleration due to gravity, m/s/s.

      // Declare/define the masses of the system.
      double m1 = 5000;   // The mass of the counterweight in kg.
      double m2;          // The mass of the beam in kg.
      double m3 = 100;    // The mass of the projectile in kg.

      // Define the heights of the system.
      double h = 2;   // The initial height of the counterweight in m.
      double h0 = 1.5;    // The height of the fulcrum in m.

      // Declare the lengths of the system.
      double r1 = 0.51;         // The length of the "short" arm in m.
      double r2 = 0.91;          // The length of the "long" arm in m.
      double r3 = 0.58;          // The length of the sling in m.

      // Declare the parameters that depend on the lengths.
      double r = (r2-r1)/2;
      double Izz = m2*pow((r1+r2),2)/12+m2*pow(r,2);

      // Define the maximum time and the time increment.
      double tmax = 10;         // The maximum time in s.
      double dt = tmax/1000;    // The time increment in s.
```

```
      // Define the dimension of future vectors.
      int dim = (int)(tmax/dt+1);

      // Declare and define the time vector.
      double t[dim];
      t[0] = 0;
      for(int k=1; k<dim; k++)
      {
         t[k]=t[k-1]+dt;
      }

      // Declare the generalized coordinates and time derivatives.
      double phi1[dim];    // phi1
      double phi2[dim];    // phi2
      double phi1d[dim];   // first time derivative of phi1
      double phi2d[dim];   // first time derivative of phi2
      double phi1dd[dim];  // second time derivative of phi1
      double phi2dd[dim];  // second time derivative of phi2

      // Determine the mass of the beam.
      m2 = (7850)*pow(0.10,2)*(r1+r2);

      // Determine the initial value of phi1.
      phi1[0] = acos((h-h0)/r1);

      // Define the initial value of phi2.
      phi2[0] = 0;

      // Define the initial values of phi1d and phi2d.
      phi1d[0] = 0;
      phi2d[0] = 0;

      // Determine the initial values of phi1dd and phi2dd.
      phi1dd[0] = ( g*sin(phi1[0])*(m1*r1-m2*r-m3*r2) + m3*r2*(
g*sin(phi2[0])*cos(phi1[0]-phi2[0]) - r3*pow(phi2d[0],2)*sin(phi1[0]-
phi2[0]) - r2*pow(phi1d[0],2)*cos(phi1[0]-phi2[0])*sin(phi1[0]-
phi2[0]) ) ) / ( m1*pow(r1,2) + m3*pow(r2,2)*pow(sin(phi1[0]-
phi2[0]),2) + Izz );

      phi2dd[0] = ( r2*( pow(phi1d[0],2)*sin(phi1[0]-phi2[0]) -
phi1dd[0]*cos(phi1[0]-phi2[0]) ) - g*sin(phi2[0]) ) / r3;

      // Determine subsequent values of coordinates/derivatives.
      for(int k=1; k<dim; k++)
```

```
      {
         // Approximate with first-order Taylor Series.
         phi1[k] = phi1[k-1] + phi1d[k-1]*dt;
         phi2[k] = phi2[k-1] + phi2d[k-1]*dt;
         phi1d[k] = phi1d[k-1] + phi1dd[k-1]*dt;
         phi2d[k] = phi2d[k-1] + phi2dd[k-1]*dt;

         // Calculate phi1dd and phi2dd based on above values.
         phi1dd[k] = ( g*sin(phi1[k])*(m1*r1-m2*r-m3*r2) + m3*r2*(
g*sin(phi2[k])*cos(phi1[k]-phi2[k]) - r3*pow(phi2d[k],2)*sin(phi1[k]-
phi2[k]) - r2*pow(phi1d[k],2)*cos(phi1[k]-phi2[k])*sin(phi1[k]-
phi2[k]) ) ) / ( m1*pow(r1,2) + m3*pow(r2,2)*pow(sin(phi1[k]-
phi2[k]),2) + Izz );

         phi2dd[k] = ( r2*( pow(phi1d[k],2)*sin(phi1[k]-phi2[k]) -
phi1dd[k]*cos(phi1[k]-phi2[k]) ) - g*sin(phi2[k]) ) / r3;
      }

      // Output the values of t, phi1, and phi2 to a text file.
      ofstream file1;
      file1.open("data.txt");

      file1 << "t phi1 phi2" << endl;

      for(int k=0; k<=dim; k++)
      {
         file1 << t[k] << " " << phi1[k] << " " << phi2[k] << endl;
      }
  }
```

Appendix C: Data Generated by "graph_data.cpp"

t	phi1	phi2
0	0.198355	0
0.01	0.198355	0
0.02	0.198715	-0.000555201
0.03	0.199437	-0.00166556
0.04	0.200521	-0.00333081
0.05	0.201967	-0.0055505
0.06	0.203777	-0.00832393
0.07	0.205955	-0.0116502
0.08	0.208502	-0.015528
0.09	0.211423	-0.019956
0.1	0.214721	-0.0249324
0.11	0.218402	-0.030455
0.12	0.22247	-0.0365214
0.13	0.226932	-0.0431286
0.14	0.231794	-0.0502731
0.15	0.237064	-0.0579511
0.16	0.242749	-0.0661579
0.17	0.248858	-0.0748883
0.18	0.2554	-0.0841362
0.19	0.262384	-0.0938947
0.2	0.269821	-0.104156
0.21	0.277723	-0.114911
0.22	0.2861	-0.126149
0.23	0.294965	-0.137858
0.24	0.304331	-0.150026
0.25	0.314212	-0.162638
0.26	0.324623	-0.175676
0.27	0.335577	-0.189121
0.28	0.347092	-0.202951
0.29	0.359184	-0.217143
0.3	0.371869	-0.231669
0.31	0.385167	-0.246498
0.32	0.399096	-0.261598
0.33	0.413675	-0.276929
0.34	0.428926	-0.292449
0.35	0.444868	-0.308113
0.36	0.461525	-0.323867
0.37	0.47892	-0.339656
0.38	0.497075	-0.355416
0.39	0.516017	-0.371078
0.4	0.53577	-0.386566
0.41	0.556362	-0.401798
0.42	0.57782	-0.416684
0.43	0.600174	-0.431125
0.44	0.623451	-0.445016
0.45	0.647685	-0.458241
0.46	0.672906	-0.470677
0.47	0.699148	-0.482192

0.48	0.726444	−0.492643
0.49	0.754829	−0.501877
0.5	0.78434	−0.509734
0.51	0.815013	−0.516043
0.52	0.846886	−0.52062
0.53	0.879998	−0.523278
0.54	0.914386	−0.523814
0.55	0.950089	−0.522021
0.56	0.987148	−0.517681
0.57	1.0256	−0.51057
0.58	1.06548	−0.500456
0.59	1.10683	−0.487101
0.6	1.14968	−0.470265
0.61	1.19407	−0.449702
0.62	1.24003	−0.425165
0.63	1.28758	−0.396406
0.64	1.33675	−0.363179
0.65	1.38755	−0.325241
0.66	1.44001	−0.282354
0.67	1.49411	−0.234286
0.68	1.54988	−0.180815
0.69	1.60729	−0.121727
0.7	1.66633	−0.0568223
0.71	1.72697	0.0140842
0.72	1.78918	0.0911624
0.73	1.8529	0.174564
0.74	1.91808	0.26442
0.75	1.98464	0.360842
0.76	2.05248	0.463916
0.77	2.12152	0.573703
0.78	2.19163	0.690241
0.79	2.26268	0.813538
0.8	2.33455	0.943573
0.81	2.40707	1.0803
0.82	2.48009	1.22363
0.83	2.55345	1.37347
0.84	2.62698	1.52967
0.85	2.70052	1.69204
0.86	2.7739	1.86039
0.87	2.847	2.03445
0.88	2.91967	2.21393
0.89	2.99182	2.39849
0.9	3.06337	2.5877
0.91	3.13428	2.78109

Appendix D: R Script for Plotting the Data Generated by “graph_data.cpp”

```
$ graph.R

graph_data = read.table("data.txt", header = TRUE)

plot(phi1~t, data = graph_data)

plot(phi2~t, data = graph_data)
```

References

Gere, James M., and Barry J. Goodno. *Mechanics of Materials*. Seventh Edition. Mason, Ohio: Cengage Learning, 2009.

Resnick, Robert; David Halliday; and Kenneth S. Krane. *Physics: Volume One*. Fifth Edition. New York, New York: John Wiley & Sons, Inc., 2002.

Taylor, John R. *Classical Mechanics*. Sausalito, California: University Science Books, 2005.