Table and Figure Numberings are intentionally out of order. Go with it.

**Overview and Introduction**

The main goal of the Jungle Bridge project was to predict the shape of a bridge using constrained and unconstrained optimization, specifically with the gradient descent algorithm. For the unconstrained optimization aspect, we created a bridge made up of rubber band segments that could stretch as needed, and weights were placed on the ends of the segments to link them together. For the constrained optimization aspect, we followed a very similar approach, with the only change being that the bridge segments were now made up of string (which does not stretch as much as the rubber bands do, and thus we can assume that the segment lengths don’t change).

In this project, we started off with the unconstrained optimization aspect. In order for us to use the gradient descent, we needed first to use linear regression to create an equation that would model the amount the rubber bands would stretch with respect to the weight that the rubber band would hold (and the resulting tension that it would cause). Now that we had an equation that represented that relationship, we needed to create another equation that would be used by the gradient descent algorithm to determine what the most stable shape for the bridge is. To do this, we created an equation that calculated the total potential energy of the bridge; in this case, it was adding both spring potential energy and gravitational potential energy for each segment. Then, we specified that we wanted the configuration that would have the least amount of potential energy as defined by the equation we just created (this is because, from our physics models, we know that all things in the universe naturally want to have the least amount of energy; we are assuming that there is no kinetic energy in the bridge). Finally, we ran the algorithm and it outputted the optimal shape of the bridge based on masses at the joints between the segments of the bridge, the relationship between mass held and length stretched for each of the rubber band segments, and the potential energy that the bridge would have for a given shape. For the constrained optimization aspect, we used almost the same algorithm as outlined above with the exception of a few modifications to account for our use of string instead of rubber bands. Since we used string (a material that, for our purposes, does not stretch at all), we know that its relationship between its length stretched to weight held (and its resulting tension) is that for every increase in weight held, the length stretched does not change. In addition, due to the previous relationship, we can now also model the total potential energy of the bridge as being only the gravitational potential energy per segment. This is because of the previous relationship stated and how that does not allow the segment to store any potential energy. One last thing, for our constrained optimization aspect, we also made all of the segment lengths static so it would better model the constraints of the bridge. Afterwards, we ran the gradient descent algorithm, taking everything listed above into account, and it outputted the ideal shape that would allow the bridge to be most stable.

**Methodology**

Jungle Bridge (Rubber Band) Physical Model Characteristics

The Jungle Bridge we made comprised 6 rubber bands of varying lengths and 5 weights that were used to hold the rubber bands together at the ends of each segment. The weights used varied in weight (see Table 4) along with the stiffness of the rubber band segments (see Table 2).

|  |  |
| --- | --- |
| Weight Label | Weight (g) |
| 1 (Pink) | 40 |
| 2 (Red) | 25 |
| 3 (Yellow 1) | 50 |
| 4 (Light Blue) | 50 |
| 5 (Yellow 2) | 50 |

Table 4: Weights used in the jungle bridge model vertices.

To calculate the stiffness of the rubber band segments, we hung different weights from each rubber band and measured how much each rubber band stretched. We recorded our data in Table 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Measurement 1 | Measurement 2 | Measurement 3 | Measurement 4 | Untensed Measurement |
| Rubber Band #1 - Light Green | Mass (g) | 20 | 40 | 62 | 83 | 0 |
| Stretched Length (cm) | 11.75 | 12.25 | 12.25 | 12.5 | 10.5 |
| Rubber Band #2 - Dark Blue | Mass (g) | 22 | 42 | 63 | 103 | 0 |
| Stretched Length (cm) | 8.25 | 8.25 | 8.75 | 9 | 7.25 |
| Rubber Band #3 - Orange | Mass (g) | 21 | 41 | 61 | 81 | 0 |
| Stretched Length (cm) | 8.5 | 8.75 | 8.75 | 9 | 7.25 |
| Rubber Band #4 - Rust Red | Mass (g) | 20 | 40 | 62 | 103 | 0 |
| Stretched Length (cm) | 6.25 | 6.5 | 6.5 | 6.75 | 4.75 |
| Rubber Band #5 - Dusty Red | Mass (g) | 22 | 42 | 82 | 103 | 0 |
| Stretched Length (cm) | 6.25 | 6.75 | 6.75 | 7 | 5.5 |
| Rubber Band #6 - Light Brown | Mass (g) | 20 | 42 | 62 | 83 | 0 |
| Stretched Length (cm) | 9 | 9.5 | 10 | 10.25 | 7.25 |

Table 1: Measurements used to calculate the stiffness and natural length of each rubber band used in the jungle bridge model.

Using the measurements above, we computed the stiffness and natural length of each rubber band using simple linear regression.

[Linear Regression Equation AA^T …]

In order to do this, we need to prepare two different matrices, one matrix, is comprised of the length that the rubber band stretched under different weights in one column and the number 1 in the other column.

[matrix a]

The reason matrix is set up this way is because this matrix will store the basis vectors of the data, and in this way, encoding the x values of the data. The other matrix, is a column vector with the gravitational force exerted by the mass hanging from the rubber band, calculated by using Newton’s second law.

[F = ma -> f=mg]; Matrix Y]

This process finds the sum of the squared distances between the measured points (in other words, the error in the predictions made by the line of best fit) and the line of best fit and then minimizes that sum of the squared distances to make the most accurate line of best fit based on the data. This in of itself is another form of optimization problem in which our objective is to minimize the error between the measured data and the line of best fit.

[cost func 23.8]

[Figure 2; cost func]

In the end, we get a line of best fit based on our data, which we can use along with Hooke’s law to extract the stiffness, , and natural length, , for each rubber band.

[line of best fit; fig 1]

Using that, we can easily translate the slope and y-intercept of the line to stiffness, , and natural length, , values for each rubber band.

[hooke’s law stuff]m = k; -b/m=l\_0]

After all of that, we have arrived at the following calculated values. These values will be used to characterize the behavior of the rubber bands under different weights and will be used to calculate the shape of the jungle bridge.

|  |  |  |
| --- | --- | --- |
|  | | Stiffness (N/m) | | Natural Length (m) |
| Rubber Band #1 - Light Green | 77.78105263158126 | 0.11541777188328932 |
| Rubber Band #2 - Dark Blue | 85.65925925926581 | 0.07904661016949203 |
| Rubber Band #3 - Orange | 117.59999999998126 | 0.08324999999999931 |
| Rubber Band #4 - Rust Red | 162.6800000000284 | 0.06161144578313313 |
| Rubber Band #5 - Dusty Red | 100.06315789472805 | 0.06077835051546339 |
| Rubber Band #6 - Light Brown | 47.23932203389761 | 0.08613924050632896 |

Table 2: Stiffness and Natural Length calculations based on running a linear regression algorithm on the data collected.

The physical model we built looks as follows (Figure 3), with vertex measurements in the following table (Table 3).

[Figure 3; Photo of Jungle Bridge]

|  |  |  |
| --- | --- | --- |
| Coordinate Label | Coordinate (cm) | Coordinate (cm) |
| 1 (Left Vertex) | 0 | 0 |
| 2 (Weight 1 Vertex) | 6.985 | -10.16 |
| 3 (Weight 2 Vertex) | 14.2875 | -15.5575 |
| 4 (Weight 3 Vertex) | 22.5425 | -18.0975 |
| 5 (Weight 4 Vertex) | 28.8925 | -16.1925 |
| 6 (Weight 5 Vertex) | 33.3375 | -11.1125 |
| 7 (Right Vertex) | 39.37 | 0 |

Table 3: Coordinate description of the location of the jungle bridge’s vertices based on the leftmost vertex being the origin.

Unconstrained Optimization

Before we can use gradient descent to solve this unconstrained optimization problem, we first need to define what a gradient is and how to compute it numerically. A gradient, represented by the symbol Nabla (), is an operator on a function (in other words, it does something to a function) that stores all the partial derivatives of the function in every dimension. Think of this as a container for all of these partial derivatives.

[gradient def]

What this represents with all of these partial derivatives is how the surface created by a multi-variable equation is changing in every direction. Numerically speaking, we would calculate the partial derivatives of the multi-variable function and then plug in the relevant coordinates to get how much the surface is changing in a direction. In practice, it would look similar to this given the multi-variable function and the point (1, 0) at which you are trying to find the gradient at. First, you would approximate the partial derivatives by evaluating the function at two different points, separated by a very small step size, , along one direction.

Next, you would assemble these partial derivatives into the gradient which would be as follows.

Then you would plug in the and values into the gradient of the function and solve. In this example, that would look as follows. Note: The number of variables you would use would depend on the number of variables your function has. Since this function has two variables, we only need to use two variables.

We can interpret this outcome as the surface is changing in -direction but not changing in the -direction based on the values of each partial derivative component. This matches what the surface looks like, the surface is changing if you move in the -direction, but not if you move in the -direction.

[figure of surface]

Now we can begin the gradient descent algorithm to solve this unconstrained optimization problem. This algorithm uses a nuanced property of the gradient which is that the gradient will always point in the direction of the steepest ascent. This is true because the gradient is always perpendicular to the contour lines (or equivalent) of a multi-variable function and is therefore moving the quickest away from the point at which the gradient was calculated.

The gradient descent algorithm has 5 steps, not including the number of times you would iterate through a set of steps. Before starting, you will need to define an objective function (which is a relationship between the variables that you can change in your problem and the outcome of those variables) and a step size for the algorithm (this will be explained later). First, you determine whether you want to find the minimum or maximum of the objective function you have. In the case of our jungle bridge problem, it is as follows:

[cost func]

The reason the objective function we chose is as listed above is because the most likely shape of the jungle bridge would be that of when it is most at rest (given that the left and right vertices are fixed in this problem). Therefore, the objective function that we will use (as seen above) is one that calculates the total potential energy stored by the bridge which is only gravitational and elastic potential energy. In addition, we also specifically are looking for the minimum of the objective function since the bridge will be most at rest the less potential energy it has.

Next, we will select a random point and compute the gradient at that point. After that, we will check if the gradient at that point is equal to 0. This is important because if the gradient is equal to 0 (or something very close to it), that means we have found our most optimal point for the given objective function. If the gradient is equal to 0 (or something very close to it), then that is your answer; if not, multiply the gradient vector by some step size (preferably on the smaller side so the gradient descent algorithm does not pass up the optimal point) you defined before starting the gradient descent algorithm. This step size parameter is used to regulate how much the initial point changes when it is multiplied by the previous point. After you multiply the gradient vector by the step size parameter, you will multiply that vector by the previous point output by the gradient descent algorithm (if this is the first iteration through this loop, this will be your initial guess). What you are doing in this step is essentially taking one step on the path you are creating while “going down the hill” that is the surface of the multi-variable objective function in order to find the lowest point (or the minimum) in this problem. Then, you calculate the gradient at this new point output by the gradient descent algorithm and repeat steps 3-5 as many times as you would like or until you reach the optimal point.

[Diagram of Gradient Descent Algorithm]

For this problem specifically, we followed the gradient descent algorithm with a small modification. We added the backtracking line search algorithm to adjust the step size parameter dynamically as we are running the gradient descent algorithm. This is intended to help fix the issue with the gradient descent algorithm passing up the most optimal point (or never reaching it due to a step size that is too large). This algorithm changes the step size parameter based on the size of the gradient vector at that iteration’s point. If the gradient vector is large, then the step size will also be large and vice versa.

In our implementation, we iterated through the gradient descent algorithm 1000 times using the backtracking line search algorithm. Below is a visualization of this process.

[Figure 5; Progression of Gradient Descent Algorithm Iteration Guesses]

Rope Bridge Physical Model Characteristics

Constrained Optimization

|  |  |  |
| --- | --- | --- |
| Segment Name | Coordinate Labels | Length (cm) |
| 1 | 0 to 1 | 19 |
| 2 | 1 to 2 | 19.8 |
| 3 | 2 to 3 | 15 |
| 4 | 3 to 4 | 15.7 |
| 5 | 4 to 5 | 10.7 |
| 6 | 5 to 6 | 3.5 |

Table 5: [Caption Here]

|  |  |
| --- | --- |
| Weight Label | Mass (g) |
| 1 | 26 |
| 2 | 41 |
| 3 | 25 |
| 4 | 51 |
| 5 | 41 |

Table 6: [Caption Here]

|  |  |  |
| --- | --- | --- |
| Coordinate Label | Coordinate (cm) | Coordinate (cm) |
| 0 (Left Vertex) | 0 | 0 |
| 1 (Weight 1 Vertex) | 25.7 | -12.2 |
| 2 (Weight 2 Vertex) | 44.4 | -17.1 |
| 3 (Weight 3 Vertex) | 59.6 | -16.8 |
| 4 (Weight 4 Vertex) | 75.2 | -13 |
| 5 (Weight 5 Vertex) | 84.3 | -7 |
| 6 (Right Vertex) | 91.8 | 0 |

Table 7: [Caption Here]

Properties of String: Natural Length is only length; Stiffness = infinity; etc. Measured using ruler since length won’t change for our purposes.

SECTION B - UNCONSTRAINED OPTIMIZATION PROCESS

PARTIAL

ADD CODE SNIPPETS AND EQUATIONS IN BETWEEN

Gradient Descent works like this:

1. You calculate the gradient at a specific point, this is point A.
2. Go in the direction of the gradient (which points in the direction of steepest ascent) by alpha amt. Where you end up is point B. Speaking of Alpha, we will discuss this a bit farther below on how we choose this value.
3. Check if the gradient at point B is within a small range from 0; if so, you have found your minimum of the function. In our case, we are trying to find the point where the cost function has the smallest output (a minimum).
4. If gradient at point B is not within a small range from 0, repeat steps 2-3 again until you get to the “lowest point”. You are essentially “going down a hill” that is the surface of the cost function.

Alpha Value Modification Stuff/Backline Search:

* Chooses larger step sizes when the gradient is large, chooses smaller step sizes (alpha) when gradient is small (nearing minimum). This is used to fix one of the flaws with the gradient descent algorithm, that it usually passes the minimum.
* 5. Add the new scaled gradient to get new point … repeat steps.

SECTION C

Constrained Optimization Approach to Gradient Descent

The constrained optimization approach to gradient descent is very similar to the unconstrained approach. Differences are:

* You now have constraints; you have to follow them; F\* equations; Lagarange Mult.
* F\* is the objective function with the constraints included in; constraints are now represented as equations with = no inequalities
* In this problem, that is our modified cost function and …

Answer is where gradients add up to 0 -> Linearly Dependent -> Parallel -> Minimum Point of Gradient.

Constrained vs. Unconstrained Optimization - Talked above

**Results**

Linear Regression of Rubberband - Line of best fir for rubber bands (provides k and l0 values)

Ideal Bridge Config Results - Output of Generate Shape (most stable bridge)

**Interpretation**

Physics Rationale (Bridge wants to be at rest (least amt of pot energy))

Error Sources:

* Measurement Errors

**Team Attribution Statement**