Optimization of Modified Brachistochrone Curves Subject to Polynomial Constraints

Kevin Louth

Dynamics Honors Project

May 2011

Abstract

In this project, the historical math problem of the brachistochrone curve was expounded upon by calculating the polynomials of increasing degrees that yielded the minimum time for a particle traveling along these polynomials only under the influence of gravity. The computer programs MatLab and FreeMat were used for numerical integration to find the ranges of the polynomial coefficients that minimized the traversal time. A generalized method was developed to find a range of the time-minimizing coefficients for any degree polynomial by using multidimensional arrays. Using this range of coefficients, Maple was used to find a more accurate calculation for the coefficients and traversal time. When compared to the inverted cycloid classical solution for the brachistochrone curve, it was found that as the degree of the polynomial increased, the resulting curve modeled the cycloid curve more and more closely. It was also found that as the visual separation between the polynomial curves and the cycloid curve was minimized, the time approached a minimum limit for time: that of the cycloid solution. In this project, the extensive use of math tools such as FreeMat, MatLab, and Maple has been very educational, and this is experience that will inevitably be helpful in the field of engineering.

Table of Contents

List of Tables and Figures

Experimental Data

(refer to **Appendix** for all program code)

Derivation of time equation

It is important in this project to provide a general method for calculating the time required for a particle to traverse any curve. It is simple to see that the speed of the particle is changing with respect to its y-coordinate. Since speed is distance over time, time is distance divided by speed. One can imagine that over a very short interval of distance, say 'ds,' the speed hardly changes, thus an average speed on that interval will yield an accurate approximation of the time required to traverse ds. Conservation of energy holds for this situation, thus the speed of a particle can be written in terms of the y-distance it has dropped. The length of ds can be formulated via the Pythagorean theorem, where $ds^2 = dx^2 + dy^2$. Algebraic manipulation yields that $ds = (1 + (y')^2)*dx$, where $y'=dy/dx$. Combining all these concepts and terms, the following equation for time is formulated.

$$t = \int \frac{(1 + (y')^2)}{2g(16-y_{mid})}\, dx \qquad \textbf{Eq. (E.1)}$$

Though in MatLab we will integrate time numerically by summing all the differential time elements (dt), this equation is the backbone of calculating time.

Classical Solution: The Cycloid Calculation

The first step in calculating the equation for the cycloid is to use the general parameterized equations of a cycloid to find the specific cycloid curve between the two points P=(0,16) and Q=(20,0). The general cycloid equations are as follows, where C_1 and C_2 are constants that shift the curve:

$x = r(\theta - \sin(\theta)) + C_x$

$y = r(\theta - \cos(\theta)) + C_y$

These parameters are both in terms of variable θ. For simplicity, the initial value of θ will be taken to be zero. Knowing that the curve plotted between A and B is not a standard cycloid, x and y coordinates must be shifted accordingly. At θ_0=0,

Using:	$x(0)=0$ and $y(0)=16$
Calculated:	$C_x=0$ and $C_y=16$

Then, Maple was used to calculate θ_f and r. At θ_f :

Using: $x(\theta_f)=20$ and $y(\theta_f)=0$

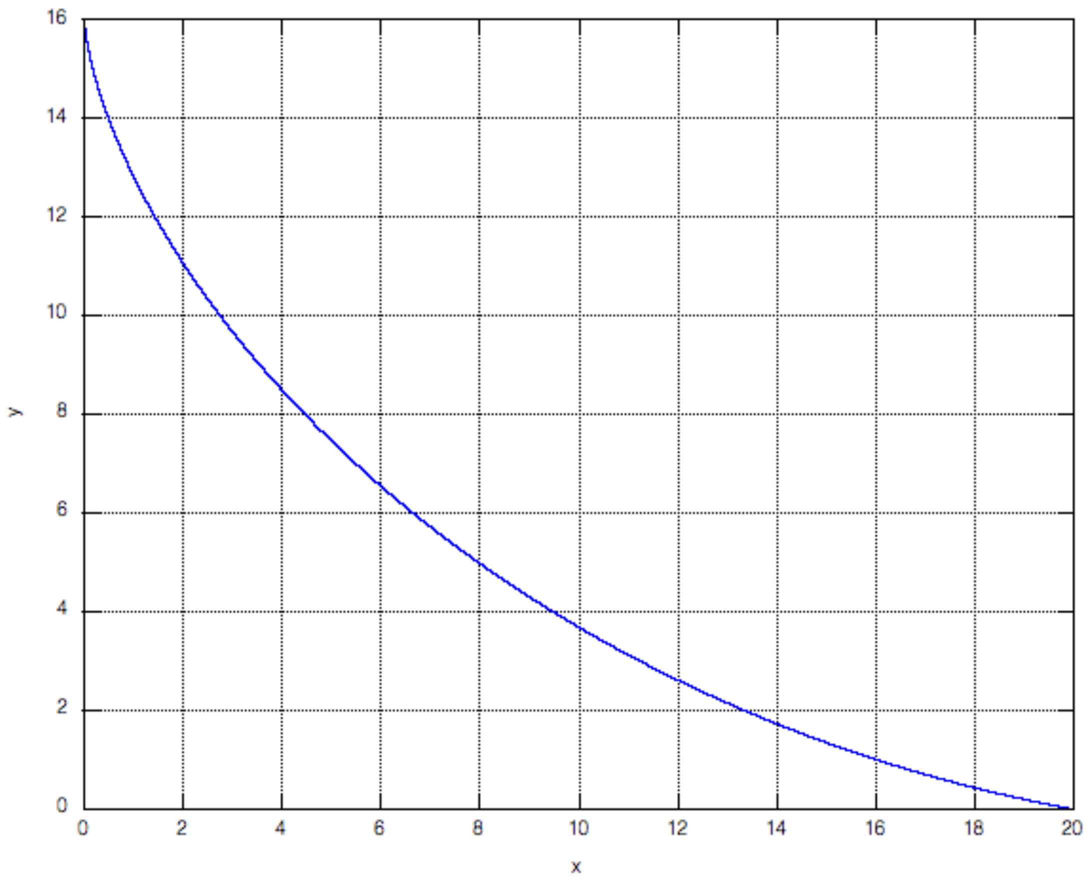
Calculated: $\theta_f = -2.775255217$ and $r = -8.274525895$

MatLab was then used to calculate the time it takes for a ball to roll down a curve. Because Maple was unable to integrate the time equation for a more accurate cycloid time, the step size of θ was increased dramatically in MatLab to yield a more accurate time. Below is the result.

$t_{cyc}= 2.54881629$ (s)

The second part of the program graphed the resulting curve.

Figure G.1: Cycloid



Polynomial Degree n=1 Calculation

There are numerous ways to calculate time for a degree n=1 polynomial, because this is a straight line and there is only one equation of a line that passes through both points P and Q. The most accurate way by far requires no integration and only involves a free body diagram of a particle.

Figure G.2: Free Body Diagram

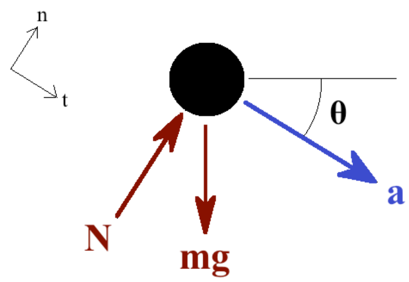
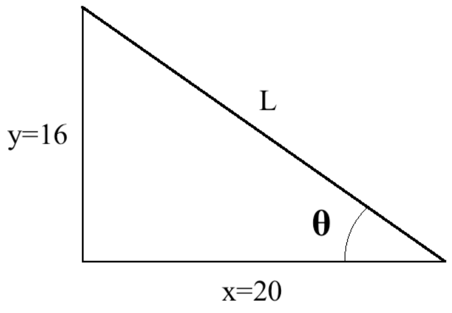


Figure G.3: Physics Trigonometry



Via trigonometry of Figure 3:

$L = (x^2 + y^2)^{0.5}$ and $\theta = \tan^{-1}(y/x)$

Since the acceleration of the particle is the tangential component of gravity:

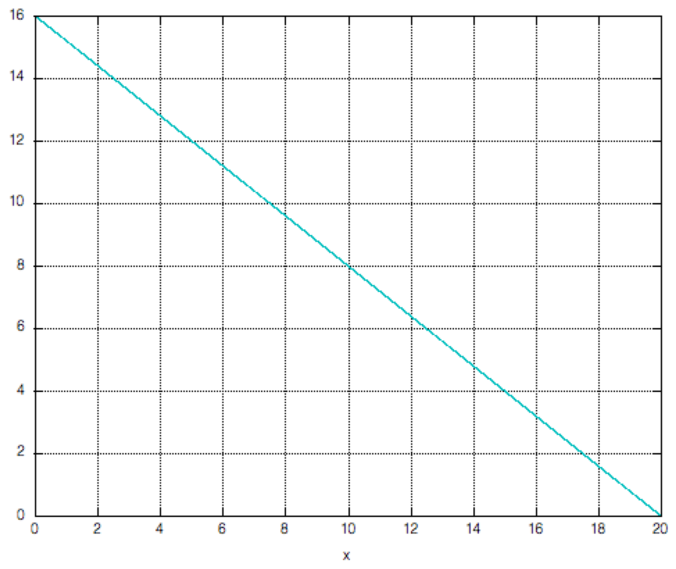
$a = g\cdot\sin(\theta)$

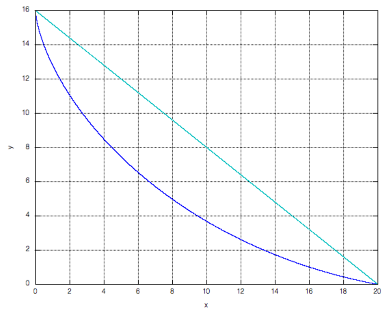
Using these, it is known that in time t, the particle initially at rest travels a distance $L = 0.5a\cdot t^2$. Solving for t and plugging in values:

$t_1 = 2.89116197$ (s)

The graph of the n=1 polynomial is of course quite easy to predict.

Figure G.4: Polynomial Degree n=1





Polynomial Degree n=2 Calculation

There are infinitely many solutions for a polynomial of degree n=2 that pass through P and Q, but only one of those solutions yields the minimum time. The basic equation for degree n=2 is below.

$y_2(x) = Ax^2 + Bx + C$

Using the initial conditions $y(x_0=0)=16$ and $y(x_f=20)=0$, y can be written in terms of A and x, where:

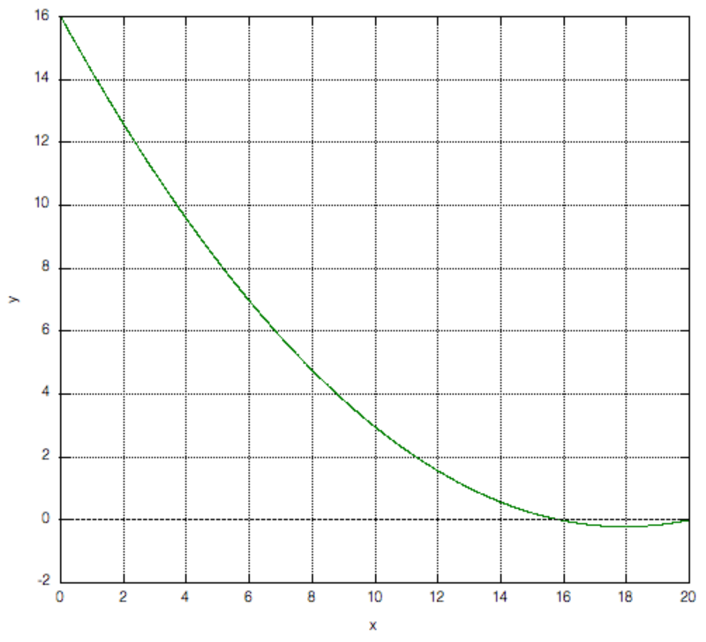
$y_2(x,A) = Ax^2 - (20A+4/5)x + 16$

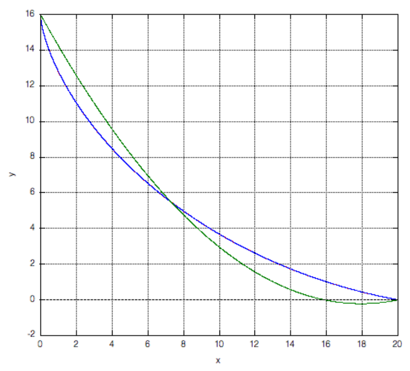
Writing a MatLab program with two vectors (x and A), one can find the A that yields the minimum time. MatLab was then used to graph the polynomial, as changes in the value of A are miniscule, and changes to the plot of the graph are undetectable to the human eye. Using the approximate value of A that has been calculated, Maple can then be used to find a more accurate minimum time value by using a guess and check approach for A.

$t_2= 2.612202573$ (s)

$y_2(x) = 0.049985x^2 - 1.7997x +16$

Figure G.5: Polynomial Degree n=2





Polynomial Degree n=3 Calculation

Like the degree n=2 polynomial, there are infinitely many solutions for the n=3 polynomial that pass through points P and Q, but only one set of coefficients that yield the minimum time. The general equation for n=3 is:

$y_3(x) = Ax^3 + Bx^2 + Cx + D$

Using initial conditions like the degree n=2 case, two of the coefficients can be written in terms of the other coefficients, thus:

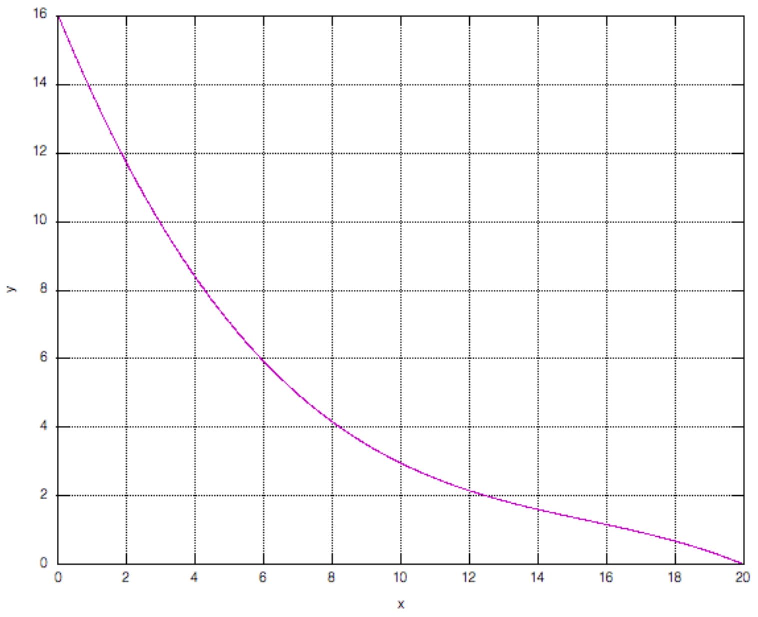
$y_3(x,A,B) = Ax^3 + Bx^2 – (400A+20B+4/5)x + 16$

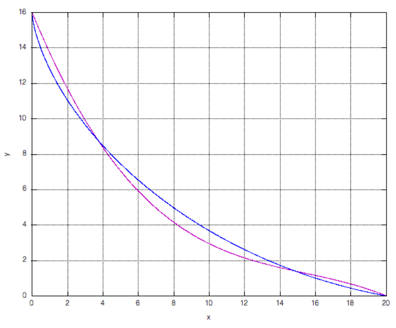
Since there are multiple coefficients to consider as variables, finding the minimizing coefficients is not as direct as for n=2. In MatLab, two possibilities come to mind: either loops or multivariable arrays. Because multidimensional arrays are faster to calculate and more difficult to visualize/set up, I of course accepted the challenge and used arrays. An explanation of how my programs work is in the following section, **Explanation of Program Logic**. Again, taking the rough coefficient values from MatLab and using Maple for fine-tuning, these are the results:

t_3= 2.578271120 (s)

$y_3(x) = -0.00311x^3 + 0.1434x^2 – 2.424x + 16$

<u>Figure G.6: Polynomial Degree n=3</u>





Polynomial Degree n=4 Calculation

Though not required to calculate a polynomial of degree n=4, I decided to test my generalized method for calculating minimizing coefficients and time to see if the method was indeed effective for higher order polynomials. Since multidimensional arrays take up a large amount of memory and the number of calculations performed by the computer increases exponentially with added dimensions, the range for each coefficient had to be kept small with a large step size so that the program could run in a reasonable time. As before, starting with the general equation of a degree n=4 polynomial then putting 2 of the coefficients in terms of the others via the initial conditions, these are the results:

$$y_4(x) = Ax^4 + Bx^3 + Cx^2 + Dx + E$$

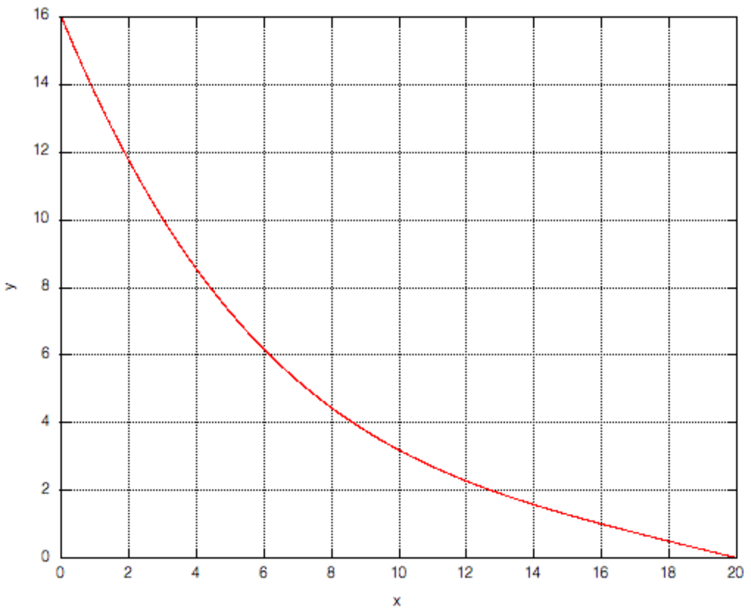
$$y_4(x,A,B,C) = Ax^4 + Bx^3 + Cx^2 - (8000A + 400B + 20C + 4/5)x + 16$$

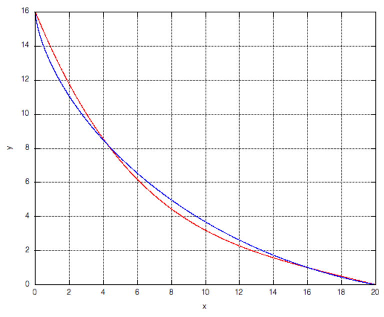
Running the program, narrowing in on the coefficients' ranges, and using Maple to find more accurate values, below are the results:

$$t_4 = 2.573758065 \text{ (s)}$$

$$y_4(x) = (6.30\cdot10^{-5})x^4 - (5.22\cdot10^{-3})x^3 + (0.616)x^2 - (2.436)x + 16$$

Figure G.7: Polynomial Degree n=4





Hypothetical Degree n=5 Calculation

Though using the general method of multidimensional arrays for degree n=5 yielded legitimate results (i.e. not an inverted polynomial, imaginary time, ridiculous numbers, etc.), I was unable to use the guess and check method with Maple to zero in on the minimizing coefficients. Though MatLab yielded approximate results, more complex programming is needed to converge on the absolute minimum time and the coefficients that yield that time. Using Maple, I was unable to calculate a time that was smaller than the time for the previous polynomials, and it would be inaccurate to use MatLab's results because of the error in numerical integration. Thus, calculating polynomial degree n=5 is possible, but requires additional methodology.

Explanation of Program Logic

The MatLab programs written to calculate the polynomials and times in the **Experimental Results** section are radically different than the classical approach of using loops for multiple variables. The use of multidimensional arrays, though not very memory efficient, takes much less time than running a program of embedded 'for' loops.

I developed a general method for calculating the approximate values for minimizing coefficients of any degree polynomial. Each of the programs begins by defining vectors for each variable in the polynomial (i.e. the variable 'x' and the free coefficients, A, B, etc.) Each of these one-dimensional vectors is defined along its own dimension, each independent of one another. Then, the length and size of each vector are assigned to variables for later reference. The next step is the fundamental idea behind this method of arrays. Each of the one-dimensional vectors is repeated in every dimension *except* the dimension of that vector. For instance, if a vector is along the x-direction and there are 3 dimensions, the vector would be repeated along the y- and z-directions, so that the vector is constant in all directions except the one in which it was originally defined. Once the vectors are repeated, they become multidimensional arrays. The size of every array is the same, and the length of each of the vectors originally defined determines the length of each dimension in the array. For instance, if the 'A' vector was 101 elements long and it was in the y-direction, the y-dimension of each array would be 101 elements long. This is the purpose of referencing each vector's length.

Next is the actual calculation of time. Since all the variables' arrays are the same size, algebraic operations can be done element by element. After defining the polynomial 'y,' the derivative of y must then be defined. While the 'diff' command might be useful, it is difficult to reference a specific dimension, so the derivative of y (y') is defined manually by finding the difference of adjacent y values divided by the difference of adjacent x values. It is important that these calculations take place in the dimension of x, as we are not differentiating with respect to coefficients. After y' is defined, we must write a modified equation for differential time (dt), using **Eq. (E.1)** referenced in **Experimental Results**. Notice that when differentiating a vector in MatLab, the length of the vector decreases by one element because differentiating finds the difference in adjacent elements.

So, the dx term of **Eq. (1)** is one element shorter than y. Therefore, we can account for this by taking y to be the midpoint between adjacent y values (adding adjacent values and dividing by two). Like differentiating, this decreases the length of y by one element, and it also makes the integration more accurate, as using a midpoint yields a much better estimate than left point or right point numerical integration. After defining dt, all the elements in dt must be summed along the x direction and stored in a new array we will call t. This decreases the number of dimensions in the dt array by one, leaving only the dimensions belonging to the coefficients. The meaning is, every value inside the t array is a time value, and the index numbers for each of these elements correspond to the index numbers for the corresponding dimension's coefficient. By finding the minimum value of t in this array, then recording and calling the index numbers that reference this minimum value, one can locate the unique set of coefficients that minimize time. After defining a new one-dimensional vector for x, one can use the minimizing coefficients to make a new one-dimensional vector for y in terms of x, and then make a plot of y versus x.

After successfully running this program, one should manually decrease the range and decrease the step size of the one-dimensional vectors for each coefficient according to the values calculated. Supposing the program is written correctly, this should be the only change necessary to converge on more accurate values for the minimizing coefficients. To ensure your time is accurate, use a reasonably small step size for the x vector. Make sure not to change this x vector when adjusting the coefficient ranges, as x is a true variable, and altering it will give you invalid results.

Discussion

It makes sense that as the polynomials increased in degree, they modeled the cycloid solution more and more closely. It is highly doubtful that any function with restrictions imposed on it (i.e. a polynomial of restricted degree) would turn out to be a better solution than the cycloid solution theorized by Leibnitz, Newton, etc. Since trigonometric functions can be written as polynomials tending towards degree $n=\infty$, it makes sense that since the parameterizations for cycloid curves involve trigonometric functions, and if indeed a cycloid is the best solution to the brachistochrone problem, then as the degree of the polynomial modeling the cycloid tends to ∞, then the difference in the cycloid solution and the polynomial graphically tends to zero. Hypothetically, if I would have successfully calculated polynomial degree $n=5$ or higher, the polynomial would have nearly masked the cycloid curve just by shear similarity.

Looking at the programming that went into this project, I have yet to hypothesize a faster way, or developed an alternative method, to calculating the time of a polynomial and the corresponding minimizing coefficients. Multidimensional arrays seem like a very effective way to approach a problem of several variables when solving said problem numerically.

Conclusions

- The higher the degree of polynomial, the more closely it models the cycloid brachistochrone curve between two given points.
- As polynomial degree increases, the minimum time yielded from the polynomial curve decreases to an asymptotic limit. The results of this project indicate that this limit is the time it takes for a particle to traverse the inverted cycloid (the classical brachistochrone solution).
- Substituting Taylor series polynomials for the trigonometric functions in the cycloid parameters (e.g. $\cos(\theta)$ and $\sin(\theta)$) theoretically yields the same minimum time as the cycloid. This is true so long as the degree of the Taylor series polynomial tends towards ∞ (infinity).
- My intelligence is not of the same caliber as that of Isaac Newton, but perseverance on this project led me to successfully calculate the requested polynomials and corresponding times.

Appendix

(MatLab program coding)

P.0 (cycloid)

```
% z=theta
clear all

cx = 0;
cy = 16;
r = -8.274525895;
zf = -2.775255217;
z = [0:-0.00001:zf]';
M = ones(size(z));

x = r*(z-sin(z)); % + cx
y = r*(M-cos(z))+cy*M;

plot(x,y,'b');
axis equal;
grid on;

g = 9.81;
n = length(z);
ymid = (y(1:n-1)+y(2:n))/2;
yp = diff(y)./diff(x);
N = ones(size(yp));
dx = (x(2:n)-x(1:n-1));
dt = (sqrt((1+yp.^2)./(2*g*(16-ymid)))) .* dx;

m = length(dt);
t = sum(dt(1:m))
```

P.1 (polynomial degree n=1)

```
% physical calculation
clear all
x = [0:0.001:20];
y = (-4/5)*x + 16;

h = 16;
b = 20;
L = sqrt(h^2 + b^2);
theta = atan(h/b);
g = 9.81;
gx = g*sin(theta);
t = sqrt(2*L/gx);
t

plot(x,y,'g');
axis equal;
grid on;
```

OR

```
% numerical calculation
clear
g = 9.81;
m = -4/5;
b = 16;
x = [0:0.0001:20];
y = m*x + b;

plot(x,y,'c');
axis equal;
grid on;

n = length(y);
ypM = diff(y)./diff(x);
yp = ypM(1);
ymid = (y(1:n-1)+y(2:n))/2;
dx = (x(2:n)-x(1:n-1));
dt = sqrt((1+yp^2)/(2*g*(16-ymid))) .* dx;

m = length(dt);
t = sum(dt(1:m))

plot(x,y,'r');
axis equal;
grid on;
```

P.2 (polynomial degree n=2)

```
clear all
A = [0.0502:0.000001:0.0504]; % A = row vector
B = -20*A-4/5; % B = row vector
C = 16;

x = [0:.01:20]'; % x = column vector
y = (x.^2)*A + x*B + C;

nA = length(A);
nx = length(x);
g = 9.81;
x1 = x*ones(size(A));
yp = diff(y)./diff(x1);
ymid = (y(2:nx,1:nA) + y(1:nx-1,1:nA))/2;
dx = (x1(2:nx,1:nA)-x1(1:nx-1,1:nA));
dt = sqrt(( 1 + yp.^2) ./ (2*g*(C-ymid))) .* dx;

choose =  195;

% A = [0:.01:2], n=6 yields t_min; A(6)=0.05
% A = [0.04:0.0001:0.06], n=104; A(104)=0.0503
% A = [0.0502:0.000001:0.0504], n=190:200….
% A = utter overkill precision

L = length(dt);
t = sum(dt(1:L,choose));

Amin = A(choose);
B = -20*Amin - 4/5;
C = 16;
y = Amin*x.^2 + B*x + C;

axisX = [0 20];
axisY = [0 0];
plot (x,y,'c',axisX,axisY,'k--');
axis equal;
grid on;
printf('t = %.4f  A = %.4f  B = %.4f  C = %.0f\n',
            t,Amin,B,C);
```

P.3 (polynomial degree n=3)

```
clear all

x(1,1,:) = [0:0.01:20]; % extrusion
A(1,:,1) = [-0.0032:0.00005:0.0030]; % row
B(:,1,1) = [0.142:0.0005:0.144]; % column

lA = length(A);
lB = length(B);
lx = length(x);
sA = size(A);
sB = size(B);
sx = size(x);

% C = -(400*A + 20*B + 4/5);
D = 16;

x = repmat(x, [lB,lA,1]);
A = repmat(A, [lB,1,lx]);
B = repmat(B, [1,lA,lx]);

%% three 3-D arrays

y = A.*x.^3+B.*x.^2+(-(400*A+20*B+4/5)).*x +D;
yp = (y(:,:,2:lx)-y(:,:,1:lx-1))./(x(:,:,2:lx)-x(:,:,1:lx-1));
dt = sqrt((1+yp.^2)./(2*9.81*( D - (y(:,:,1:lx-1)
            +y(:,:,2:lx))/2   ))) .* (x(:,:,2:lx)-x(:,:,1:lx-
1));
t = sum(dt,3);
[tmin J] = min(min(t));
[tmin I] = min(min(t'));

Amin = A(I,J,1);
Bmin = B(I,J,1);
Cmin = -(400*Amin + 20*Bmin + 4/5);
x_p = [0:0.01:20];
y_p = Amin*x_p.^3 + Bmin*x_p.^2 + Cmin*x_p + D;

axisX = [0 20];
axisY = [0 0];
tmin = tmin
printf('A = %.5f   B = %.5f   C = %.5f   D = %.0f\n',
            Amin, Bmin, Cmin, D);
plot(x_p,y_p,'m',axisX,axisY,'k--');
axis equal;
grid on;
```

P.4 (polynomial degree n=4)

```
clear all

x(1,1,1,:) = [0:0.05:20]; % block number
A(1,1,:,1) = [0.00005:0.000001:0.00007]; % depth #
B(1,:,1,1) = [-0.0052:0.00001:-0.0050]; % column #
C(:,1,1,1) = [0.158:0.0001:0.160]; % row #

lA = length(A);
lB = length(B);
lC = length(C);
lx = length(x);
sA = size(A);
sB = size(B);
sC = size(C);
sx = size(x);

% D = -8000*A - 400*B - 20*C - 4/5;
E = 16;

x = repmat(x, [lC,lB,lA,1]);
A = repmat(A, [lC,lB,1,lx]);
B = repmat(B, [lC,1,lA,lx]);
C = repmat(C, [1,lB,lA,lx]);

%% four 4-D arrays

y = A.*x.^4 + B.*x.^3 + C.*x.^2
      - (8000*A + 400*B + 20*C + 4/5).*x + E;
yp = (y(:,:,:,2:lx)-y(:,:,:,1:lx-1))
         ./ (x(:,:,:,2:lx)-x(:,:,:,1:lx-1));
dt = sqrt((1+yp.^2)
              ./(2*9.81*(E-(y(:,:,:,1:lx-1)+y(:,:,:,2:lx))/2)))
       .* (x(:,:,:,2:lx)-x(:,:,:,1:lx-1));
t = sum(dt,4);

[tmin K] = min(min(min(t)));
[NA1 J] = min(min(t(:,:,K)));
[NA2 I] = min(t(:,J,K));

Amin = A(1,1,K,1);
Bmin = B(1,J,1,1);
Cmin = C(I,1,1,1);
Dmin = -8000*Amin - 400*Bmin - 20*Cmin - 4/5;

x_plot = [0:0.01:20];
y_plot = Amin*x_plot.^4 + Bmin*x_plot.^3 +
Cmin*x_plot.^2 + Dmin*x_plot + E;

axisX = [0 20];
axisY = [0 0];
tmin = tmin
printf('A=%.7f  B=%.6f  C=%.5f  D=%.5f  E=%.0f\n',
            Amin,Bmin,Cmin,Dmin,E);
```

P.5 (polynomial degree n=5)

```
clear all

x(1,1,1,1,:) = [0:0.01:20]; % block column #
A(1,1,1,:,1) = [-2e-7:1e-8:-1e-7]; % block row #
B(1,1,:,1,1) = [-3.9e-5:1e-6:-3.7e-5]; % depth #
C(1,:,1,1,1) = [-4.1e-4:1e-6:-3.9e-4]; % column #
D(:,1,1,1,1) = [0.0908:0.0001:0.0912]; % row #

lA = length(A);
lB = length(B);
lC = length(C);
lD = length(D);
lx = length(x);
sA = size(A);
sB = size(B);
sC = size(C);
sD = size(D);
sx = size(x);

% E = -(160000*A + 8000*B + 400*C + 20*D + 4/5);
F = 16;

x = repmat(x, [lD,lC,lB,lA,1]);
A = repmat(A, [lD,lC,lB,1,lx]);
B = repmat(B, [lD,lC,1,lA,lx]);
C = repmat(C, [lD,1,lB,lA,lx]);
D = repmat(D, [1,lC,lB,lA,lx]);

%% five 5-D arrays

y = A.*x.^5 + B.*x.^4 + C.*x.^3 + D.*x.^2 -
(160000*A
     + 8000*B + 400*C + 20*D + 4/5).*x + F;
yp = (y(:,:,:,:,2:lx)-y(:,:,:,:,1:lx-1))
         ./ (x(:,:,:,:,2:lx)-x(:,:,:,:,1:lx-1));
dt = sqrt((1+yp.^2)./(2*9.81*( F - (y(:,:,:,:,1:lx-1)
             +y(:,:,:,:,2:lx))/2   )))
       .* (x(:,:,:,:,2:lx)-x(:,:,:,:,1:lx-1));
t = sum(dt,5);

[tmin N] = min(min(min(min(t))));
[NA1 K] = min(min(min(t(:,:,:,N))));
[NA2 J] = min(min(t(:,:,K,N)));
[NA3 I] = min(t(:,J,K,N));

Amin = A(1,1,1,N,1);
Bmin = B(1,1,K,1,1);
Cmin = C(1,J,1,1,1);
Dmin = D(I,1,1,1,1);
Emin = -(160000*Amin + 8000*Bmin + 400*Cmin
             + 20*Dmin + 4/5);
```

```
plot(x_plot,y_plot,'r',axisX,axisY,'k--');
axis equal;
grid on;
```

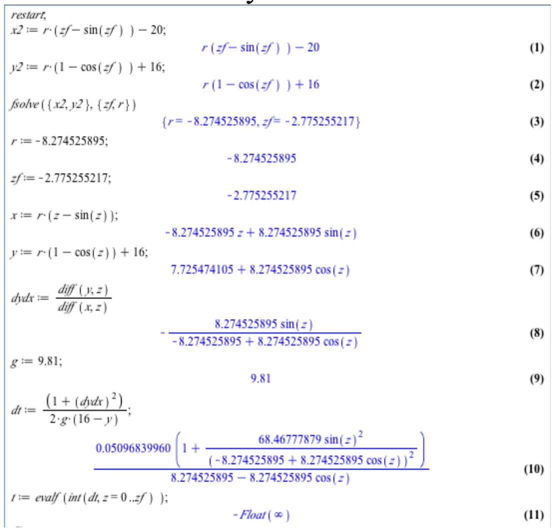
```
x_plot = [0:0.01:20];
y_plot = Amin*x_plot.^5 + Bmin*x_plot.^4 +
Cmin*x_plot.^3 + Dmin*x_plot.^2 + Emin*x_plot + F;

axisX = [0 20];
axisY = [0 0];
tmin = tmin

printf('A = %.4e   B = %.4e   C = %.4e   D = %.5f
            E = %.5f   F = %d\n',
            Amin,Bmin,Cmin,Dmin,Emin,F);
plot(x_plot,y_plot,'k',axisX,axisY,'k--');
axis equal;
grid on;
```

Maple Program Coding

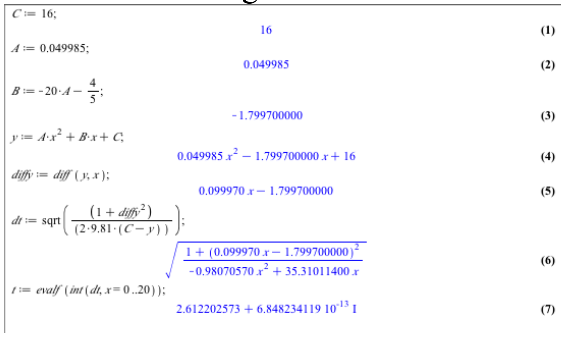
Cycloid



$restart,$

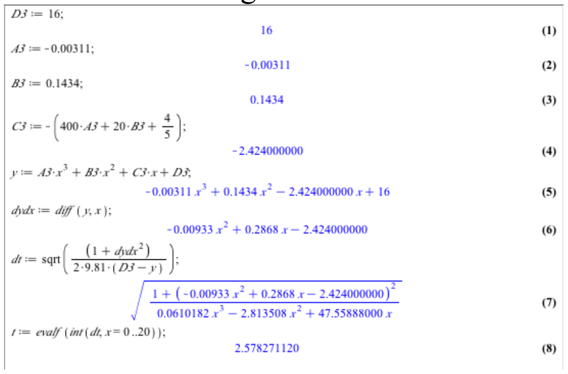
$x2 := r\cdot(zf - \sin(zf)) - 20;$
$$r(zf - \sin(zf)) - 20 \tag{1}$$
$y2 := r\cdot(1-\cos(zf)) + 16;$
$$r(1-\cos(zf)) + 16 \tag{2}$$
$fsolve(\{x2, y2\}, \{zf, r\})$
$$\{r = -8.274525895, zf = -2.775255217\} \tag{3}$$
$r := -8.274525895;$
$$-8.274525895 \tag{4}$$
$zf := -2.775255217;$
$$-2.775255217 \tag{5}$$
$x := r\cdot(z - \sin(z));$
$$-8.274525895\,z + 8.274525895\sin(z) \tag{6}$$
$y := r\cdot(1-\cos(z)) + 16;$
$$7.725474105 + 8.274525895\cos(z) \tag{7}$$
$dydx := \frac{diff(y,z)}{diff(x,z)}$
$$-\frac{8.274525895\sin(z)}{-8.274525895 + 8.274525895\cos(z)} \tag{8}$$
$g := 9.81;$
$$9.81 \tag{9}$$
$dt := \frac{(1+(dydx)^2)}{2\cdot g\cdot(16-y)};$
$$\frac{0.05096839960\left(1 + \frac{68.46777879\sin(z)^2}{(-8.274525895 + 8.274525895\cos(z))^2}\right)}{8.274525895 - 8.274525895\cos(z)} \tag{10}$$
$t := evalf(int(dt, z = 0..zf));$
$$-Float(\infty) \tag{11}$$

Degree n=2



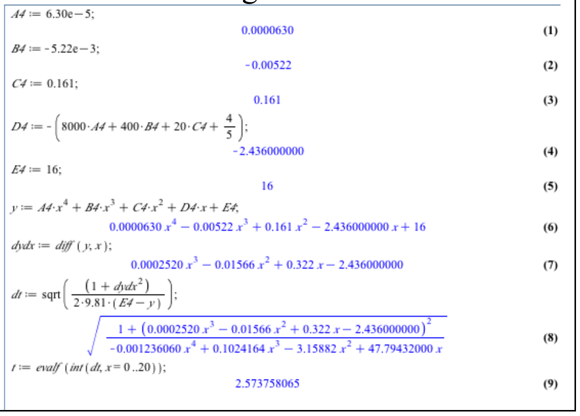
$C := 16;$
$$16 \tag{1}$$
$A := 0.049985;$
$$0.049985 \tag{2}$$
$B := -20\cdot A - \frac{4}{5};$
$$-1.799700000 \tag{3}$$
$y := A\cdot x^2 + B\cdot x + C;$
$$0.049985\,x^2 - 1.799700000\,x + 16 \tag{4}$$
$diffy := diff(y, x);$
$$0.099970\,x - 1.799700000 \tag{5}$$
$dt := \mathrm{sqrt}\left(\frac{(1 + diffy^2)}{(2\cdot 9.81\cdot(C-y))}\right);$
$$\sqrt{\frac{1 + (0.099970\,x - 1.799700000)^2}{-0.98070570\,x^2 + 35.31011400\,x}} \tag{6}$$
$t := evalf(int(dt, x = 0..20));$
$$2.612202573 + 6.848234119\;10^{-13}\,\mathrm{I} \tag{7}$$

Degree n=3



$D3 := 16;$
$$16 \tag{1}$$
$A3 := -0.00311;$
$$-0.00311 \tag{2}$$
$B3 := 0.1434;$
$$0.1434 \tag{3}$$
$C3 := -\left(400\cdot A3 + 20\cdot B3 + \frac{4}{5}\right);$
$$-2.424000000 \tag{4}$$
$y := A3\cdot x^3 + B3\cdot x^2 + C3\cdot x + D3;$
$$-0.00311\,x^3 + 0.1434\,x^2 - 2.424000000\,x + 16 \tag{5}$$
$dydx := diff(y, x);$
$$-0.00933\,x^2 + 0.2868\,x - 2.424000000 \tag{6}$$
$dt := \mathrm{sqrt}\left(\frac{(1 + dydx^2)}{2\cdot 9.81\cdot(D3-y)}\right);$
$$\sqrt{\frac{1 + (-0.00933\,x^2 + 0.2868\,x - 2.424000000)^2}{0.0610182\,x^3 - 2.813508\,x^2 + 47.55888000\,x}} \tag{7}$$
$t := evalf(int(dt, x = 0..20));$
$$2.578271120 \tag{8}$$

Degree n=4



$A4 := 6.30e-5;$
$$0.0000630 \tag{1}$$
$B4 := -5.22e-3;$
$$-0.00522 \tag{2}$$
$C4 := 0.161;$
$$0.161 \tag{3}$$
$D4 := -\left(8000\cdot A4 + 400\cdot B4 + 20\cdot C4 + \frac{4}{5}\right);$
$$-2.436000000 \tag{4}$$
$E4 := 16;$
$$16 \tag{5}$$
$y := A4\cdot x^4 + B4\cdot x^3 + C4\cdot x^2 + D4\cdot x + E4;$
$$0.0000630\,x^4 - 0.00522\,x^3 + 0.161\,x^2 - 2.436000000\,x + 16 \tag{6}$$
$dydx := diff(y, x);$
$$0.0002520\,x^3 - 0.01566\,x^2 + 0.322\,x - 2.436000000 \tag{7}$$
$dt := \mathrm{sqrt}\left(\frac{(1 + dydx^2)}{2\cdot 9.81\cdot(E4-y)}\right);$
$$\sqrt{\frac{1 + (0.0002520\,x^3 - 0.01566\,x^2 + 0.322\,x - 2.436000000)^2}{-0.001236060\,x^4 + 0.1024164\,x^3 - 3.15882\,x^2 + 47.79432000\,x}} \tag{8}$$
$t := evalf(int(dt, x = 0..20));$
$$2.573758065 \tag{9}$$