Some Calculus & Numerical Analysis

Why are we covering this topic?

- Because there were some tasks about Calculus in the regional!

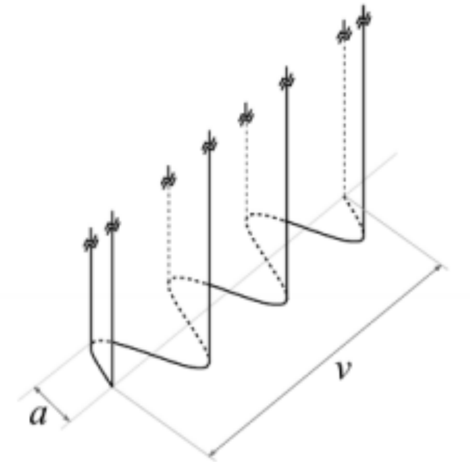


Figure 1: Curtain geometry. Note that this particular example illustrates 3 folds in a window width of v.

For each input record, output the amplitude of the sinusoidal folds required width of v using a curtain made using fabric with a of width w. The amplitud the folds form a curve in the shape $a\sin(\frac{2\pi n}{v}x)$, with $0 \le x \le v$, and $a > 0$.

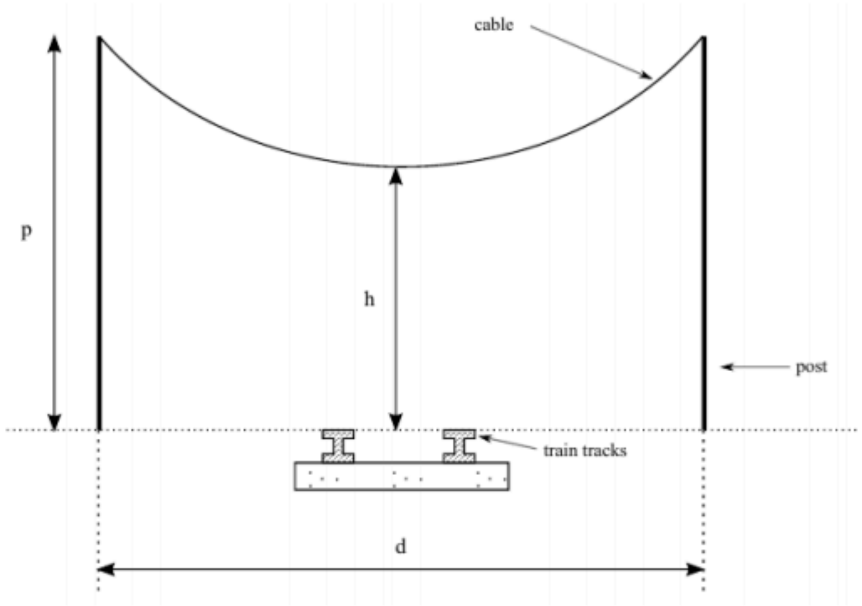


Figure 1: Power line dimensions

A cable hanging between poles is known to assume a specific shape called a *catenary*. T described by the formula

$$f(s) = a\cosh\left(\frac{s}{a}\right)$$

where $\frac{-d}{2} \le s \le \frac{d}{2}$ denotes a position along the cable, as measured on the ground, and

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

Today..

- How to use these functions in C++
  - Exponential functions & Logarithms
  - Square roots
  - Trigonometric functions
  - Hyperbolic functions
- Numerical Integration (Simpson's Rule)
- Bisection method
- Ternary search

Today..

- **How to use these functions in C++**
  - Exponential functions & Logarithms
  - Square roots
  - Trigonometric functions
  - Hyperbolic functions
- Numerical Integration (Simpson’s Rule)
- Bisection method
- Ternary search

Exponential Functions

- Exponential function with base a $(a \neq 1)$

$$f(x) = a^x$$

- To use in C++:

```
#include <math.h>

double y = pow(5.1, 7.2);
```

```
     double pow (double base     , double exponent);
      float pow (float base      , float exponent);
long double pow (long double base, long double exponent);
```

Exponential Functions (cont.)

- Caution: There are some invalid inputs.
  - $(-5)^3$ is valid, but $(-5)^{2.5}$ is *invalid. (domain error)*
  - 0^5 is valid, but 0^0 and 0^{-3} are both *invalid (domain error)*
  - $0.0013^{10000.53}$ is too small to represent by a `double`-type variable. *(range error)*
- If you call `pow` function with these values, an error occurs.
- However, although 1^x is not a exponential *function*, it has a value 1, so the `pow(1, 73.2)` returns 1.

Exponential Functions (cont.)

- e is a special constant, so $e^x = \exp x$ is also defined.
- To use in C++:

```
#include <math.h>

double x = 1.53;
double y = exp(x);
          // 4.618177
```

```
     double exp (double x);
      float exp (float x);
long double exp (long double x);
```

- If **x** is too large, e^x is super large to be represented by a variable, so an error occurs. *(range error)*

Logarithms

- e is a special constant, so $\log_e x$ is also defined.
- To use in C++:

```
#include <math.h>

double x = 51.2;
double y = log(x);
          // 3.935740
```

```
     double log (double x);
      float log (float x);
long double log (long double x);
```

- If `x` is *not positive*, an error occurs. (*domain error*)
- Logarithms with base 10 is also defined: `log10(x)`

Logarithms (cont.)

- Base conversion:

$$\log_a x = \frac{\log x}{\log a} = \texttt{log(x)/log(a)}$$

- There is no built-in function for logarithms with different bases.

Square roots

- You may use the `pow` function to calculate the square root of `x`:

```
#include <math.h>

double y = pow(5.2, 0.5); // 2.280351
```

- However, there is a built-in function for square roots:

```
#include <math.h>

double y = sqrt(5.2); // 2.280351
```

Trigonometric Functions

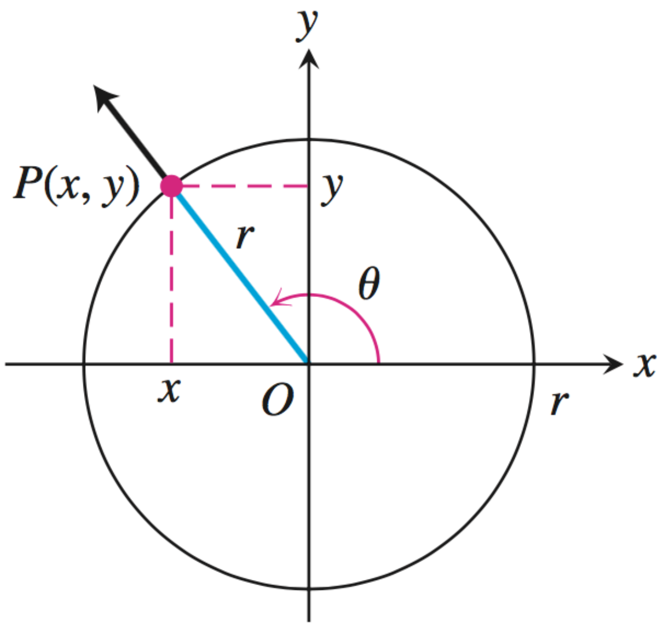


FIGURE 1.42 The trigonometric functions of a general angle θ are defined in terms of x, y, and r.

sine: $\sin\theta = \dfrac{y}{r}$ **cosecant:** $\csc\theta = \dfrac{r}{y}$

cosine: $\cos\theta = \dfrac{x}{r}$ **secant:** $\sec\theta = \dfrac{r}{x}$

tangent: $\tan\theta = \dfrac{y}{x}$ **cotangent:** $\cot\theta = \dfrac{x}{y}$

Trigonometric Functions (cont.)

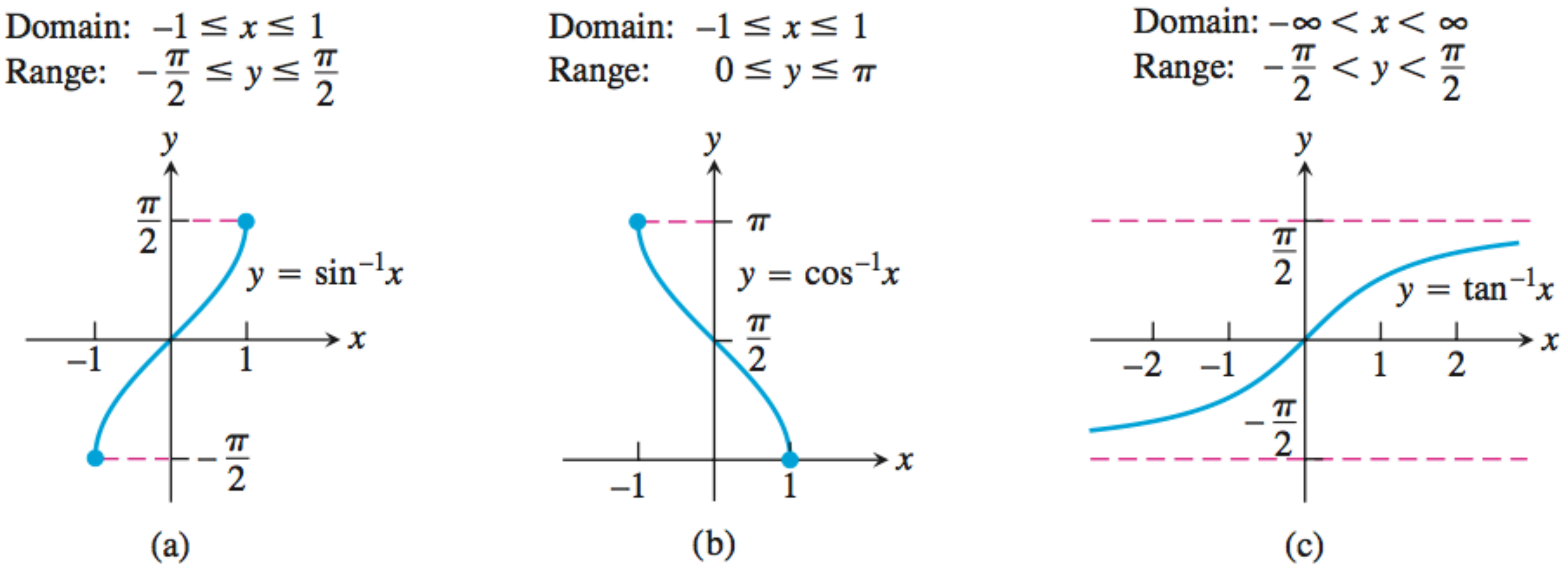
- $\sin\theta$, $\cos\theta$ and $\tan\theta$ are all defined in C++:

```
#include <math.h>

double rad = 3.14159265358979 / 6;
               // about pi / 6 = 30 degrees
double a = sin(rad); // 0.500000
double b = cos(rad); // 0.866025
double c = tan(rad); // 0.577350
```

- You <u>must</u> use <u>***radians***</u>, not degrees! $180° = \pi$ rad.
- To calculate $\csc\theta$, $\sec\theta$ and $\cot\theta$, just take the reciprocal of $\sin\theta$, $\cos\theta$ and $\tan\theta$, respectively.

Trigonometric Functions (cont.)

- What about inverses of trigonometric functions?



Trigonometric Functions (cont.)

- $\sin^{-1} x$, $\cos^{-1} x$ and $\tan^{-1} x$ are all defined in C++:

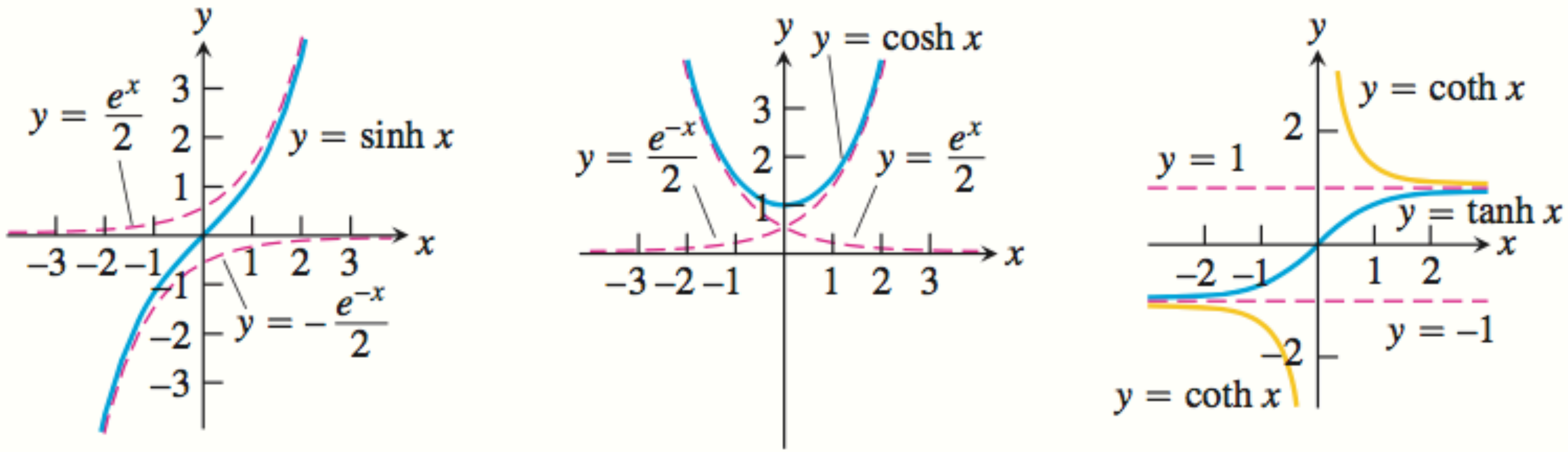
```
#include <math.h>

double x = 0.5;
double a = asin(x); // 0.523599
double b = acos(x); // 1.047198
double c = atan(x); // 0.463648
```

- Returned values are in ***radians***, not degrees! $180° = \pi$ rad.

Trigonometric functions (cont.)

- To calculate $\csc^{-1} x$, $\sec^{-1} x$ and $\cot^{-1} x$, just calculate $\sin^{-1}\frac{1}{x}$, $\cos^{-1}\frac{1}{x}$ and $\tan^{-1}\frac{1}{x}$ (maybe $\frac{\pi}{2} - \tan^{-1}\frac{1}{x}$) instead, respectively.
- Caution: You should **be careful of the *domain*** of inverse functions. If the argument is outside of the domain, an *error* occurs! (*domain error*)

Hyperbolic Functions



(a) (b) (c)

Hyperbolic sine:

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

Hyperbolic cosine:

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

Hyperbolic tangent:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Hyperbolic Functions (cont.)

- $\sinh x$, $\cosh x$ and $\tanh x$ are all defined in C++:

```
#include <math.h>

double x = 5.3;
double a = sinh(x); // 100.165909
double b = cosh(x); // 100.170901
double c = tanh(x); // 0.999950
```

- For $\sinh x$ and $\cosh x$: if `x` is too large, results are super large to be represented by a variable, so an error occurs. *(range error)*

- To calculate $\operatorname{csch} x$, $\operatorname{sech} x$ and $\coth x$, just take the reciprocal of $\sinh x$, $\cosh x$ and $\tanh x$, respectively.

Hyperbolic Functions (cont.)

- What about inverses of hyperbolic functions?

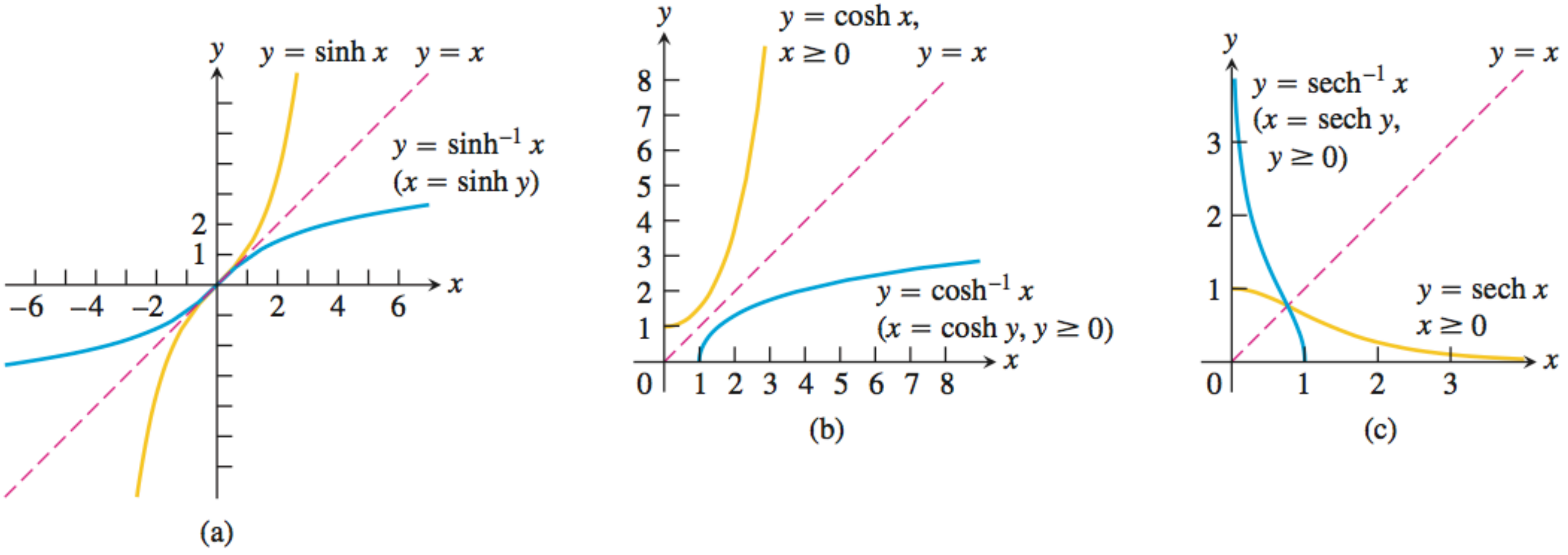


FIGURE 7.5 The graphs of the inverse hyperbolic sine, cosine, and secant of x. Notice the symmetries about the line $y = x$.

Hyperbolic Functions (cont.)

- $\sinh^{-1} x$, $\cosh^{-1} x$ and $\tanh^{-1} x$ are all defined in C++:

```
#include <math.h>
double a = asinh(0.3395); // 0.333295
double b = acosh(1.9542); // 1.290102
double c = atanh(0.8429); // 1.231107
```

- To calculate $\operatorname{csch}^{-1} x$, $\operatorname{sech}^{-1} x$ and $\coth^{-1} x$, just calculate $\sinh^{-1} \frac{1}{x}$, $\cosh^{-1} \frac{1}{x}$ and $\tanh^{-1} \frac{1}{x}$ instead, respectively.
- Caution: You should **be careful of the *domain*** of inverse functions. If the argument is outside of the domain, an *error* occurs! (*domain error*)

Today..

- How to use these functions in C++
  - Exponential functions & Logarithms
  - Square roots
  - Trigonometric functions
  - Hyperbolic functions
- **Numerical Integration (Simpson's Rule)**
- Bisection method
- Ternary search

Why integrate numerically?

- For some functions, we know how to integrate it.

$$\int e^x(\sin x + 5\cos x)dx = e^x\,(3\sin x + 2\cos x) + C$$

$$\int \frac{1}{1+x^2}dx = \tan^{-1} x + C$$

- So we can use the Fundamental Theorem of Calculus to calculate the definite integral.

$$\int_1^{\sqrt{3}} \frac{1}{1+x^2}dx = \tan^{-1} x]_1^{\sqrt{3}} = \frac{\pi}{3} - \frac{\pi}{4} = \frac{\pi}{12}$$

Why integrate numerically? (cont.)

- However, some functions are really hard to integrate:

$$\int \sqrt{1+\cos^2 x}\, dx$$

- So we just try to *approximate* the definite integral.
- For continuous functions, we can use the <u>*Simpson's Rule*</u>.

Simpson's Rule

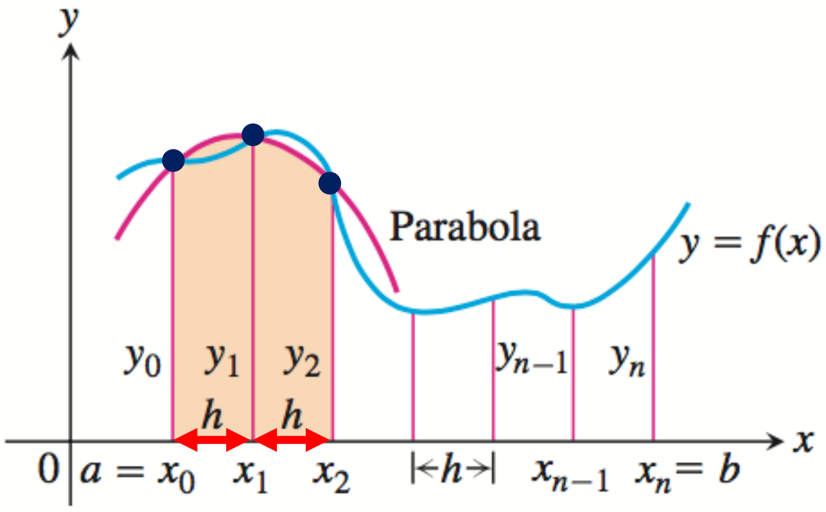
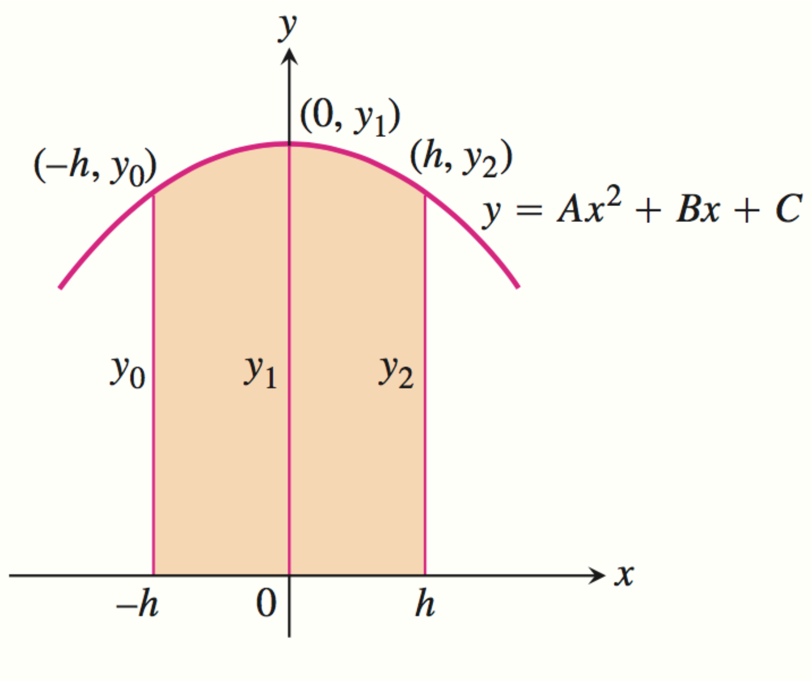


FIGURE 8.9 Simpson's Rule approximates short stretches of the curve with parabolas.

- We want to calculate $\int_a^b f(x)dx$.
- Partition the interval $[a, b]$ into n subintervals. ($h = \Delta x = (b-a)/n$)
  - n must be <u>*even*</u>.
- We ***approximate*** the curve $y = f(x)$ by pieces of **parabolas**!
  - For each consecutive pair of intervals,
  - The parabola passes three endpoints.

Simpson's Rule (cont.)



- So, we can calculate the area of the parabola instead.
- By some calculations,
  the area under the parabola
  which passes $(-h, y_0)$, $(0, y_1)$ and (h, y_2)
  from $x = -h$ to $x = h$ is:
  $$\frac{h}{3}(y_0 + 4y_1 + y_2)$$
- This formula holds even if we shift the parabola horizontally.

Simpson's Rule (cont.)

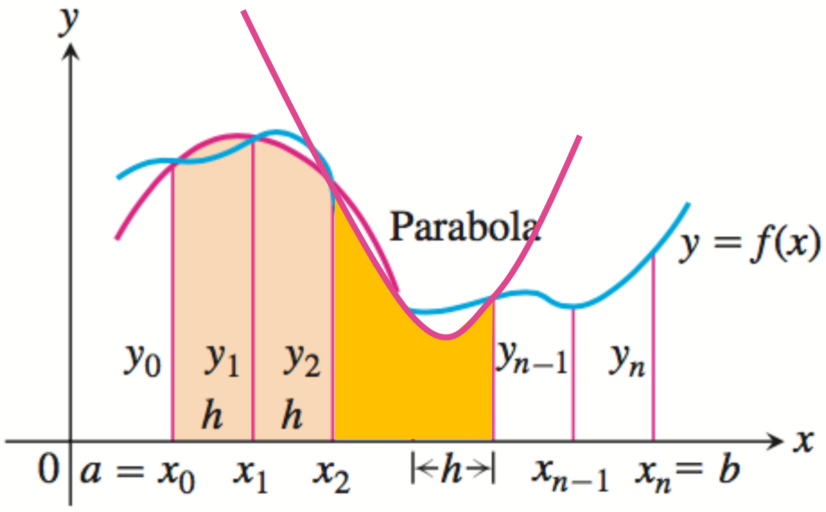


FIGURE 8.9 Simpson's Rule approximates short stretches of the curve with parabolas.

- The area under the parabola through (x_0, y_0), (x_1, y_1) and (x_2, y_2) is:
$$\frac{h}{3}(y_0 + 4y_1 + y_2)$$
- The area under the parabola through (x_2, y_2), (x_3, y_3) and (x_4, y_4) is:
$$\frac{h}{3}(y_2 + 4y_3 + y_4)$$

Simpson's Rule (cont.)

- By summing up,

$$\int_a^b f(x)dx \approx \frac{h}{3}(y_0 + 4y_1 + y_2) + \frac{h}{3}(y_2 + 4y_3 + y_4) + \cdots + \frac{h}{3}(y_{n-2} + y_{n-1} + y_n)$$
$$= \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n)$$

- f need not be positive, but it must be continuous.
- n must be even, since each parabolic arc uses two intervals.

Implementing Simpson’s Rule

```
double val = 0; // approximation of the intergral

double x_i = a;
for(int i = 0; i < n; i++) {
  double y = f(x_i);
  if(i == 0 || i == n-1) val += y;
  else if(i % 2 == 1) val += 4 * y;
  else val += 2 * y;
  x_i += h;
}

val *= h / 3; // multiply h/3 later to reduce precision error

printf("%lf\n", val);
```

1, 4, 2, 4, 2, 4, 2, 4, 1

Error Estimation in Simpson's Rule

- If $f^{(4)}$ is continuous and M is any upper bound for the values of $\left|f^{(4)}\right|$ on $[a, b]$,

$$|E_S| \leq \frac{M(b-a)^5}{180n^4}$$

- where E_S is the error in the Simpson's Rule approximation on $\int_a^b f(x)dx$ for n steps.
- As n increases, $|E_S|$ decreases very quickly.
- In most cases, it is impossible to estimate M, so just try to experiment with various n, and find the appropriate n.

Today..

- How to use these functions in C++
  - Exponential functions & Logarithms
  - Square roots
  - Trigonometric functions
  - Hyperbolic functions
- Numerical Integration (Simpson's Rule)
- **Bisection method**
- Ternary search

Bisection method

- Sometimes we want to find the roots of an equation.

$$f(x) = 0$$

- Some equations are easy to solve.

$$x^2 + 2x + 1 = 0 \;\Rightarrow\; x = -1$$

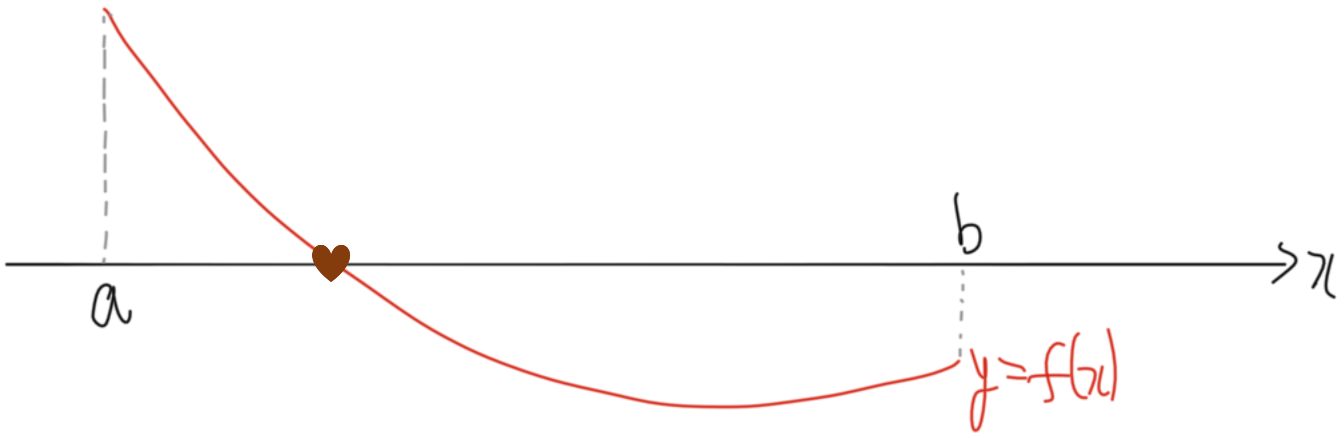
- But some are not.

$$x + \sin x + 1 = 0$$

- So we are trying to solve it numerically.

Bisection method (cont.)

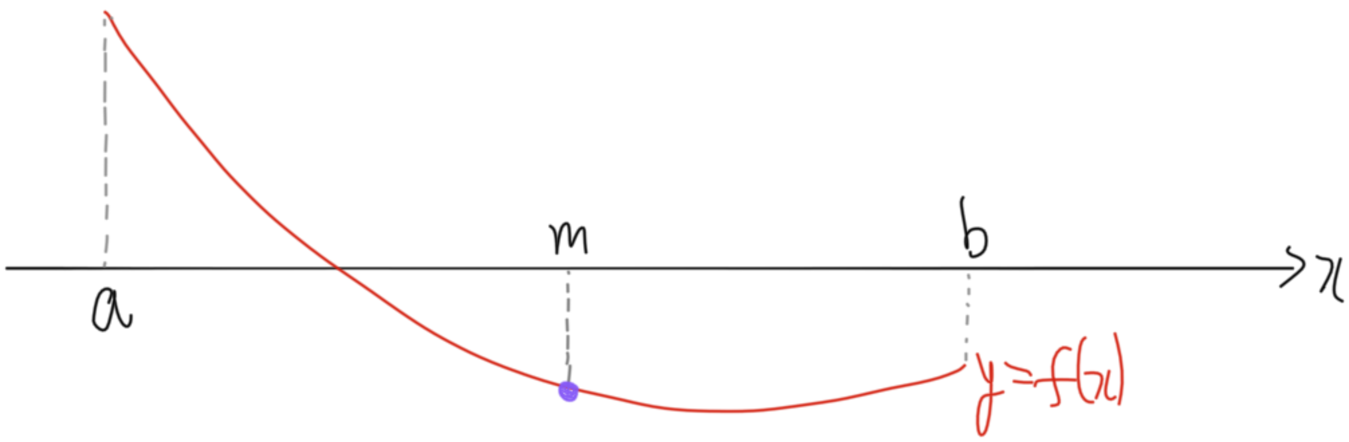
- Suppose f is a continuous function.



- When $f(a)f(b) \leq 0$, there must be *at least* one root between a and b (inclusive)
  - We are going to use this fact!
  - WLOG, $f(a) > 0$ and $f(b) < 0$.

Bisection method (cont.)

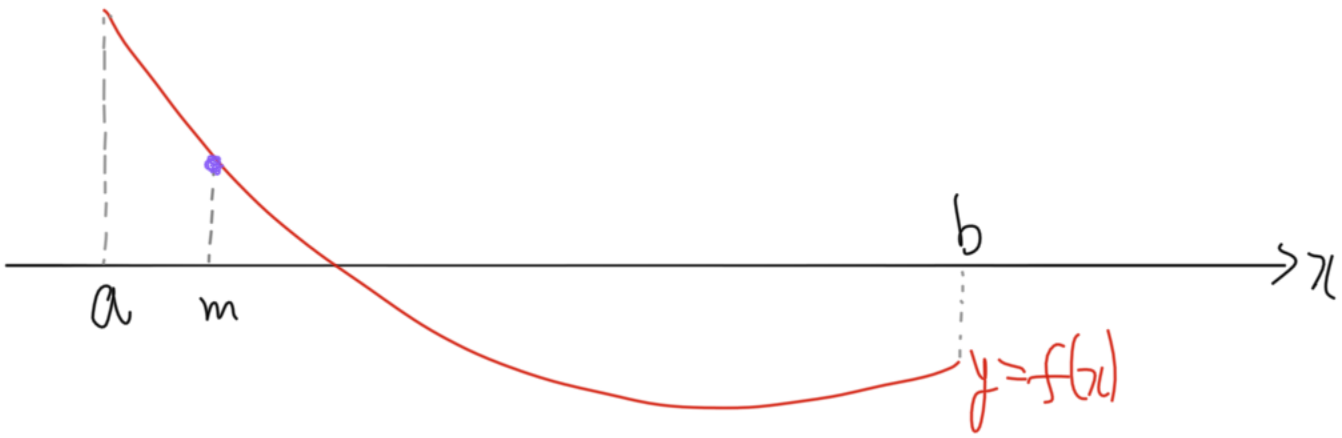
- Choose any m between a and b.



- If $f(m) < 0$, $f(a)f(m) < 0$. Therefore there exists a root between a and m.

Bisection method (cont.)

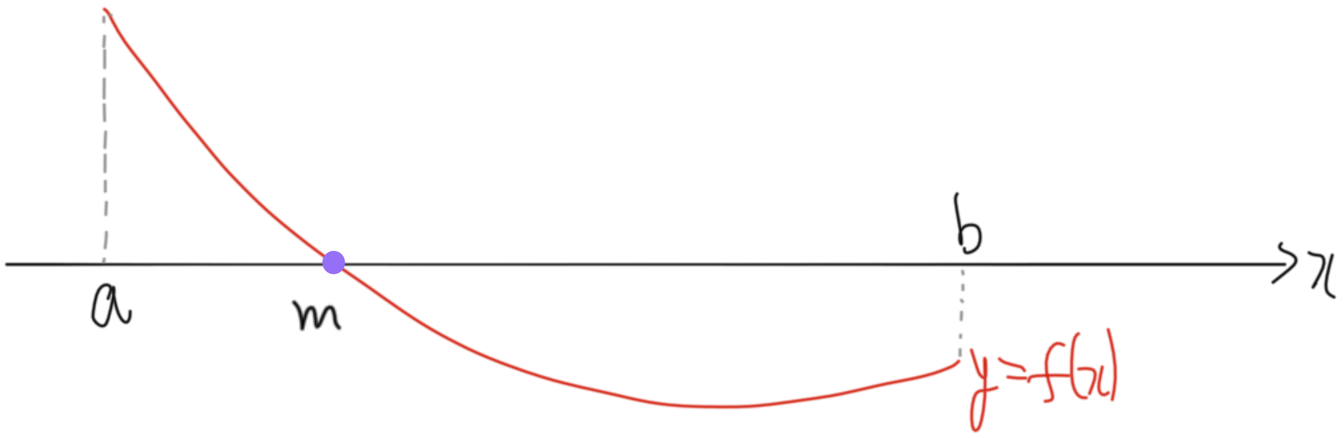
- Choose any m between a and b.



- If $f(m) > 0$, $f(b)f(m) < 0$. Therefore there exists a root between m and b.

Bisection method (cont.)

- Choose any m between a and b.



- If $f(m) = 0$, we are done.
- However, because of precision issues, this is not the case in most situations. For simplicity, let's just think there is a root between a and m.

Bisection method (cont.)

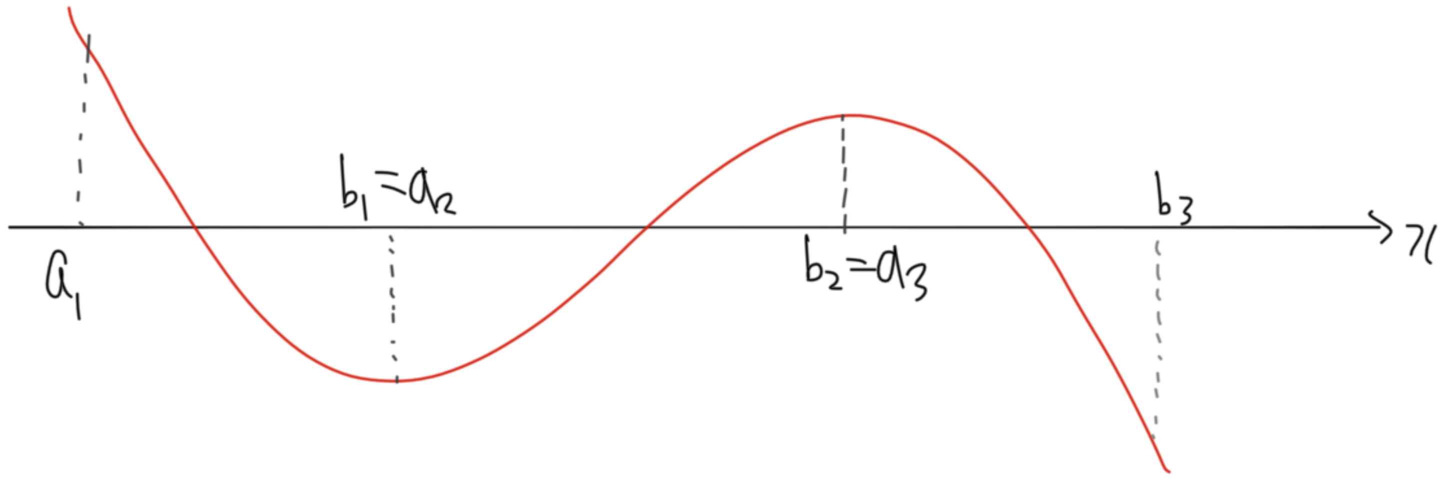
- Therefore, in both cases, the interval which contains the root shrinks.
  - $f(m) \leq 0$: $[a, b] \rightarrow [a, m]$
  - $f(m) > 0$: $[a, b] \rightarrow [m, b]$
- So choose $m = \frac{a+b}{2}$. Then, the length of the interval *always* becomes **half**.
- When we do this n times, the length of the interval becomes $(b-a) \times \left(\frac{1}{2}\right)^n$.

Bisection method (cont.)

```
// Precondition: f(a) > 0, f(b) < 0
for(int steps = 0; steps < 50; steps++) {
    double m = (a + b) / 2.0;
    if(f(m) <= 0) {
        b = m;
    }else {
        a = m;
    }
}
```

Bisection method (cont.)

- This method computes **exactly one root** between a and b. If you want to compute multiple roots, you should apply this method multiple times for different initial intervals.

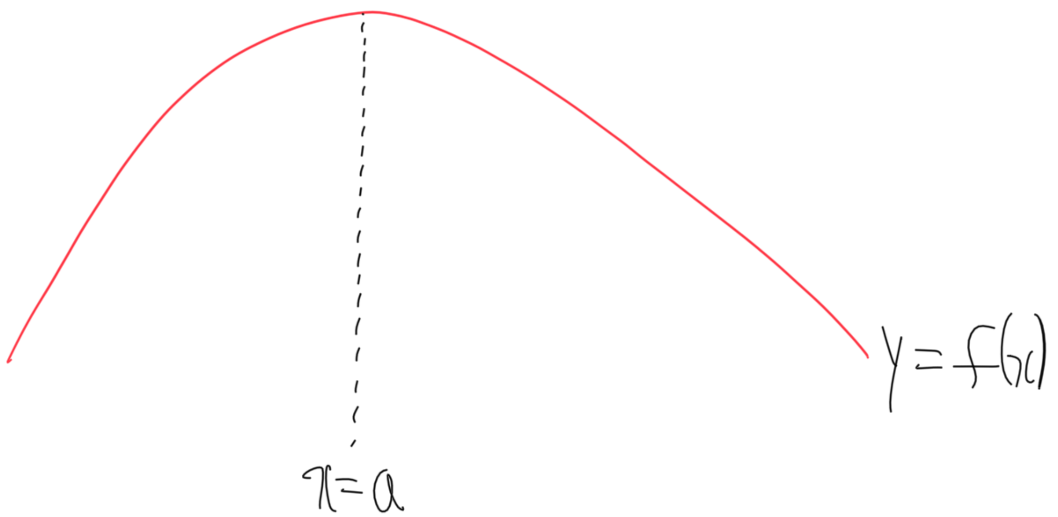


Today..

- How to use these functions in C++
  - Exponential functions & Logarithms
  - Square roots
  - Trigonometric functions
  - Hyperbolic functions
- Numerical Integration (Simpson's Rule)
- Bisection method
- **Ternary search**

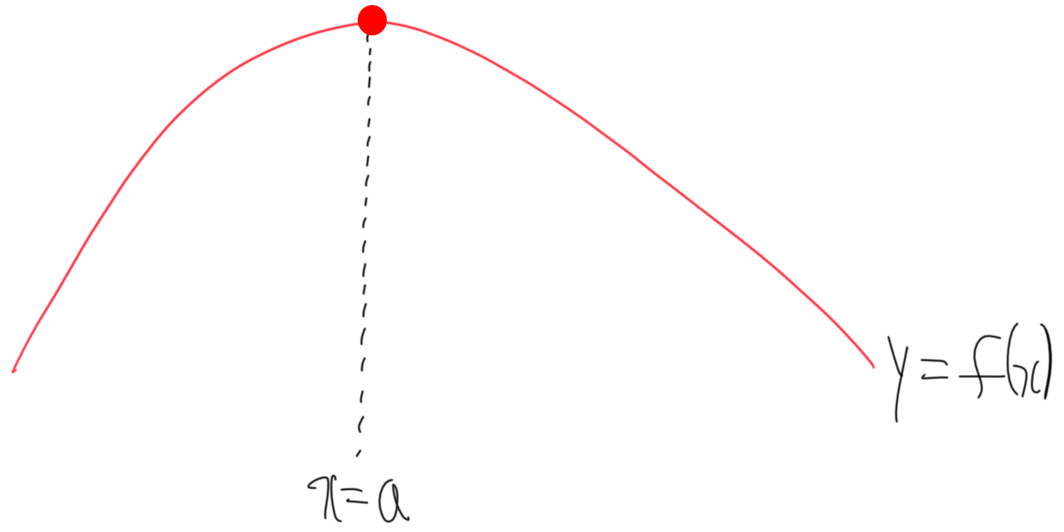
Finding the maximum of a unimodal function

- Assume we are finding the maximum of a continuous function $f(x)$.
- ..where $f(x)$ is strictly *increasing* when $x < a$ and strictly *decreasing* when $x > a$.



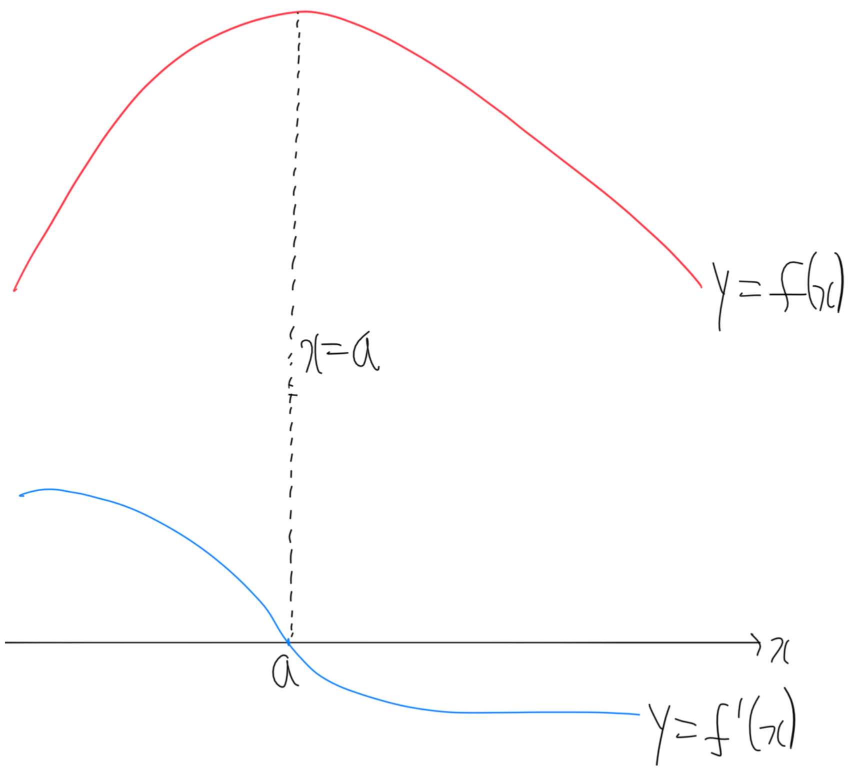
Finding the maximum of a unimodal function (cont.)

- It is obvious that the maximum value lies on $x = a$. So our task is to find the value of a.
- Of course, we don't know how the graph looks like, so we should calculate the value of $f(x)$ for some x and search for the maximum.



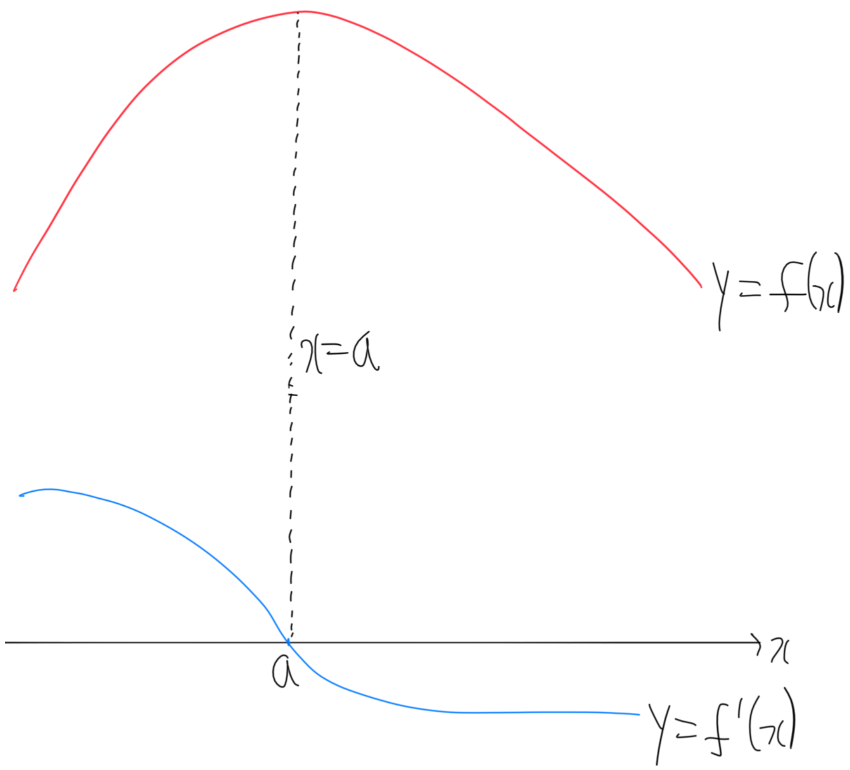
If we could calculate the derivative..

- Take a look at the derivative of f. It is positive for all $x < a$, zero if $x = a$, and is negative for all $x > a$.



If we could calculate the derivative.. (cont.)

- So we can use the *bisection method* on the derivative to find the value of a.

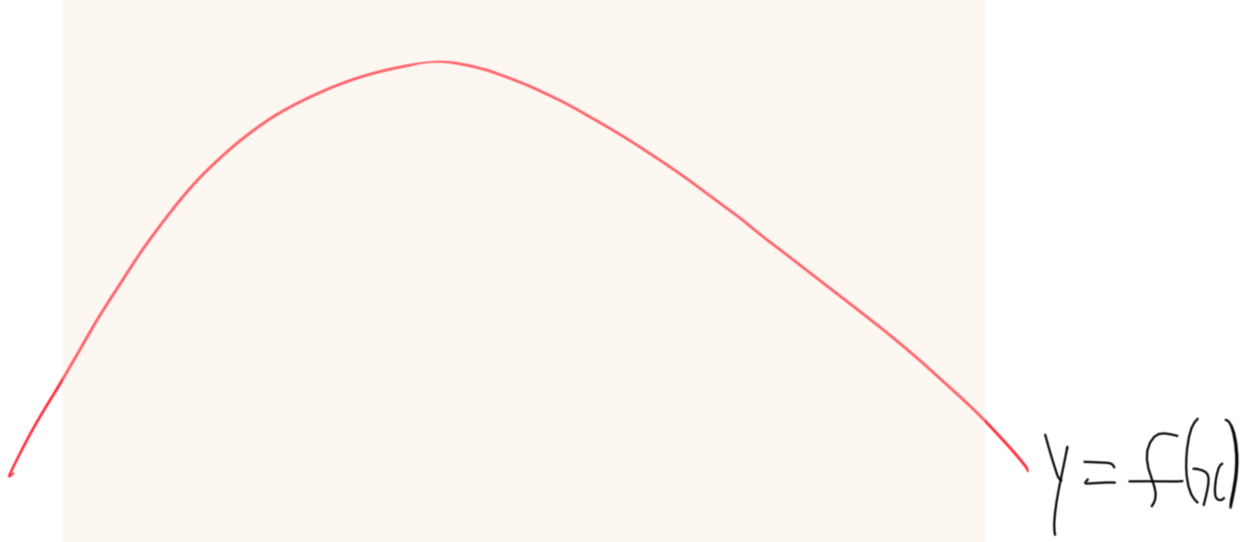


Derivative is too complicated and cause precision errors

- However, we don't know what the function f is, so we cannot calculate the value of the derivative.
- Maybe we could try to *estimate* the derivative like: $\frac{f(x+10^{-9})-f(x)}{10^{-9}}$.
- However, it isn't possible because of *precision errors*.

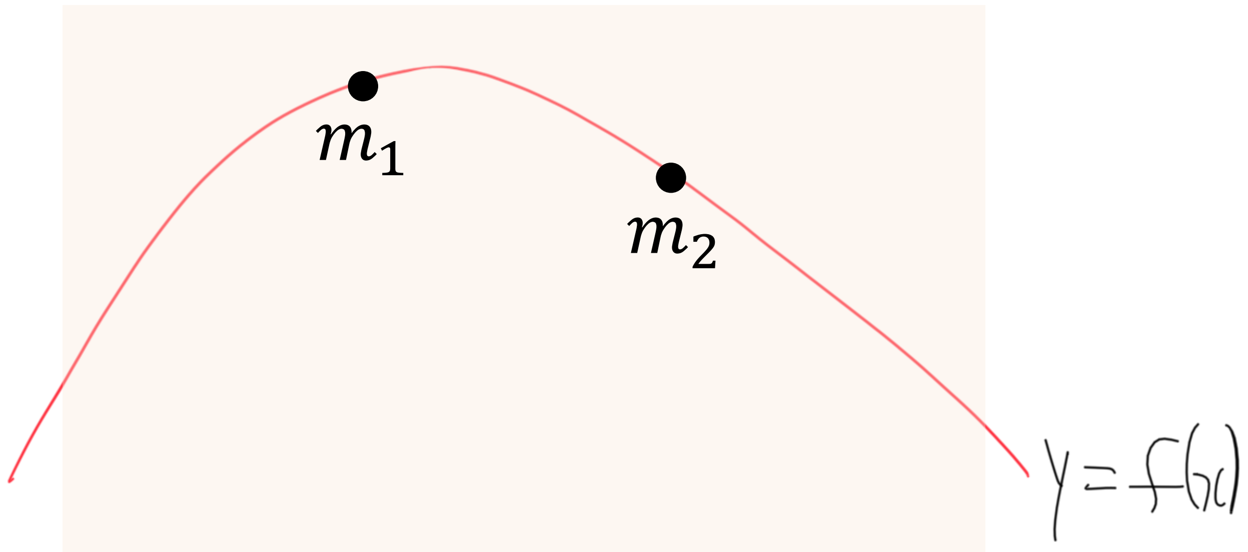
Ternary search

- This is why we use a new method, *ternary search*.
- Like other searches, first we set an interval $[l, r]$ that will contain the optimal point a.



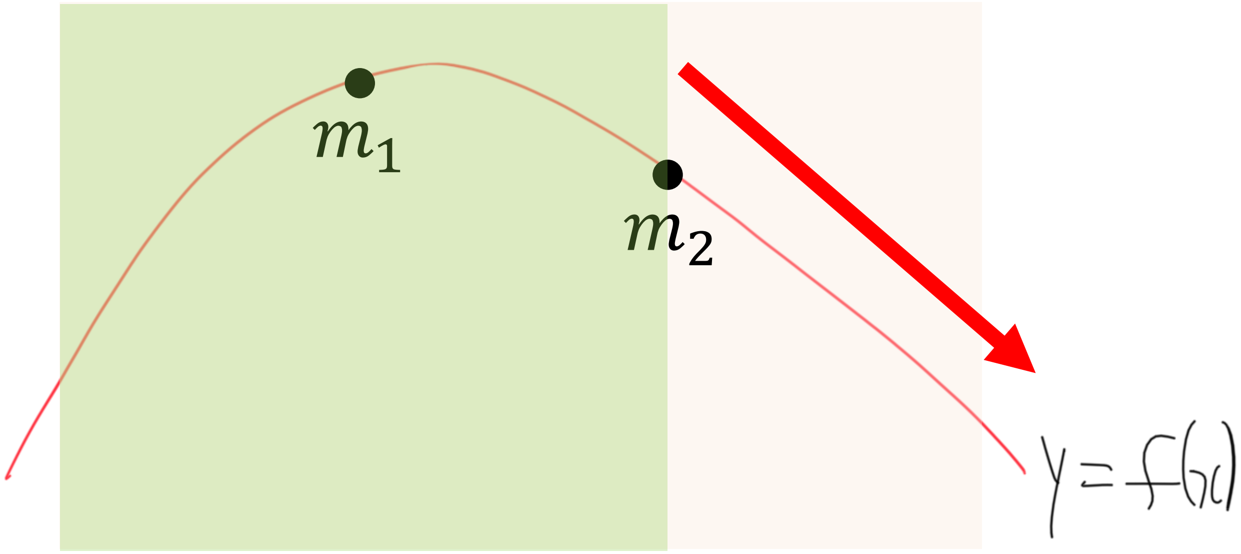
Ternary search (cont.)

- Now, consider any two points m_1 and m_2 in the interval, and calculate $f(m_1)$ and $f(m_2)$.
- What can we know from that?



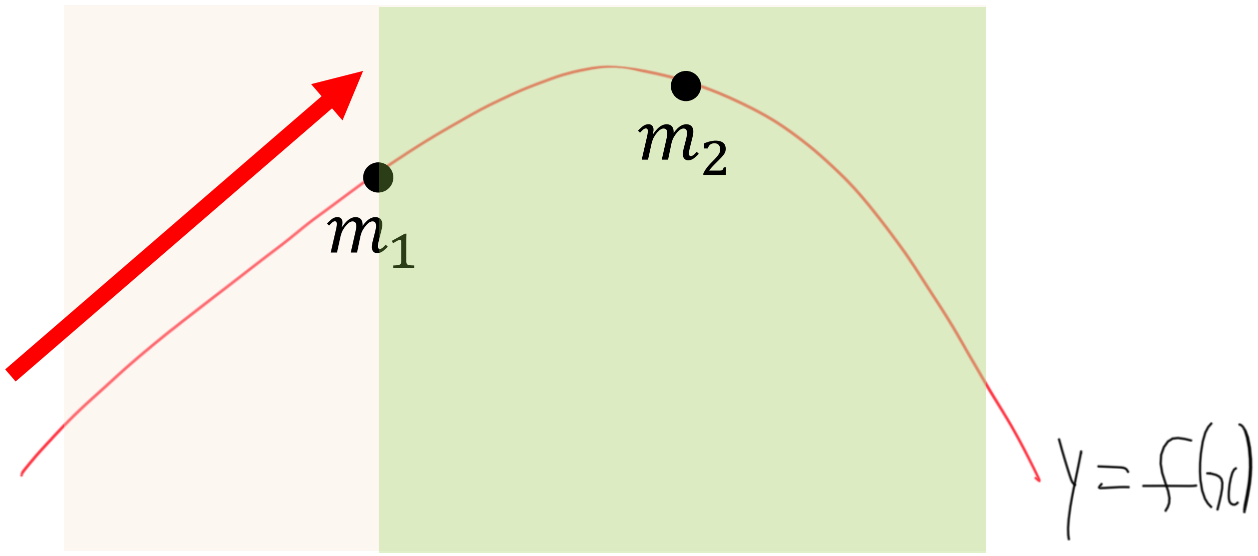
Ternary search (cont.)

- If $f(m_1) > f(m_2)$, maximum not lies in $[m_2, r]$, because f is decreasing at $x = m_2$. So we can shrink the interval to $[l, m_2]$.



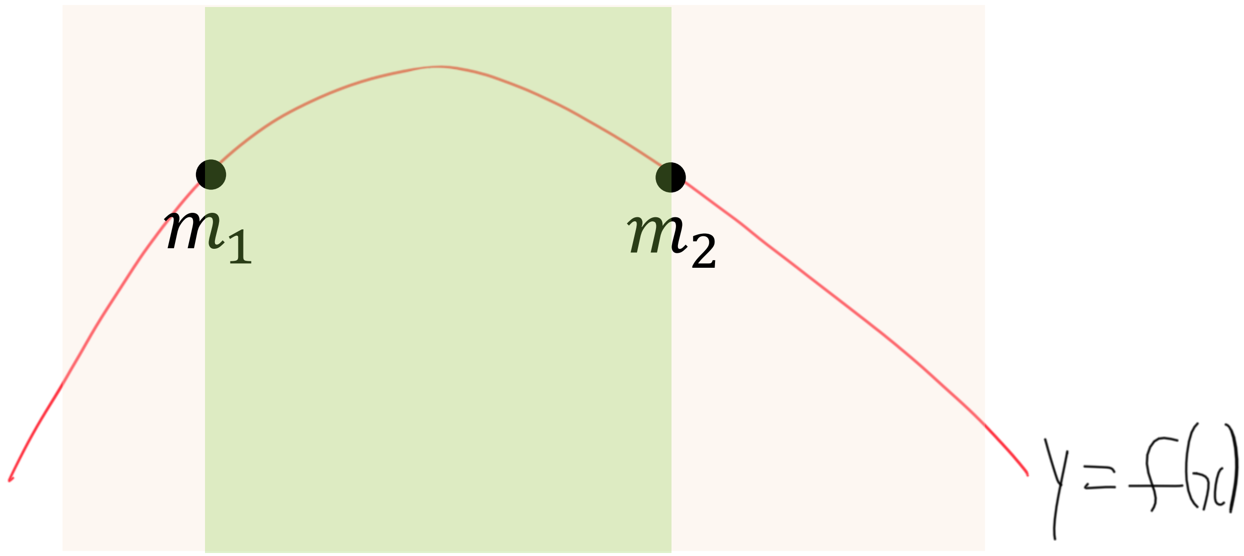
Ternary search (cont.)

- If $f(m_1) < f(m_2)$, maximum not lies in $[l, m_1]$, because f is increasing at $x = m_1$. So we can shrink the interval to $[m_1, r]$.



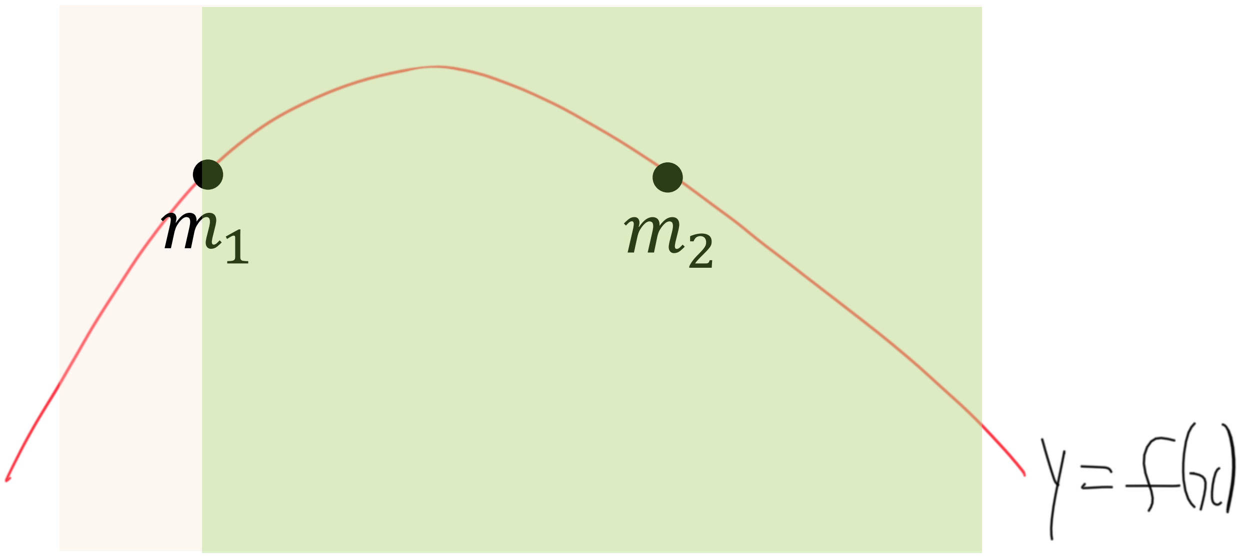
Ternary search (cont.)

- If $f(m_1) = f(m_2)$, maximum must lie in $[m_1, m_2]$, so we can shrink the interval to that.



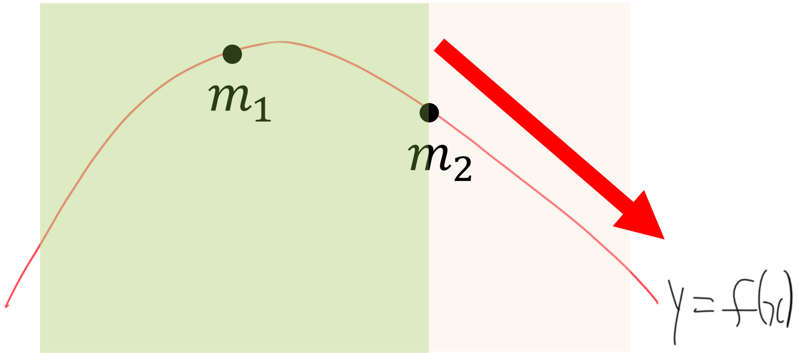
Ternary search (cont.)

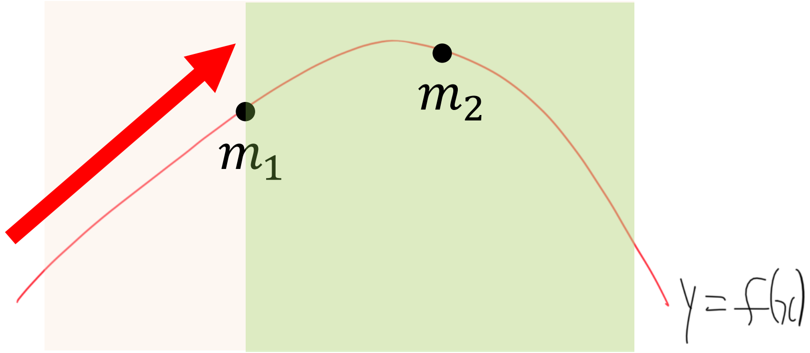
- Or you can just shrink the interval into $[l, m_2]$ or $[m_1, r]$, to make the code simpler. We will take $[m_1, r]$.



Ternary search (cont.)

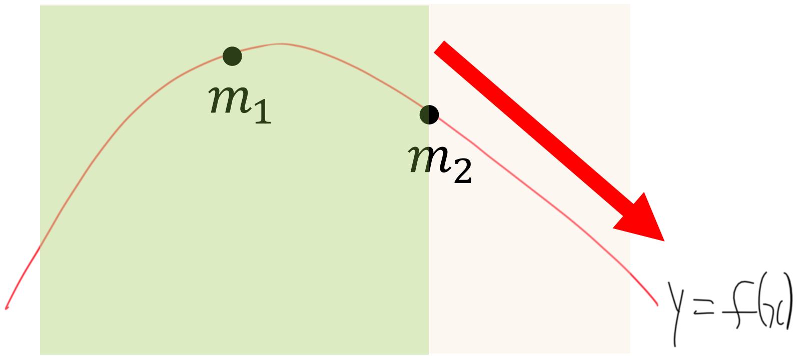
- To sum up: We have an interval $[l, r]$ that contains the optimal point $x = a$. Repeat these steps:
  - Set two points $m_1 < m_2$ in the interval.
  - If $f(m_1) > f(m_2)$, shrink the interval to $[l, m_2]$.
  - Otherwise, shrink the interval to $[m_1, r]$.

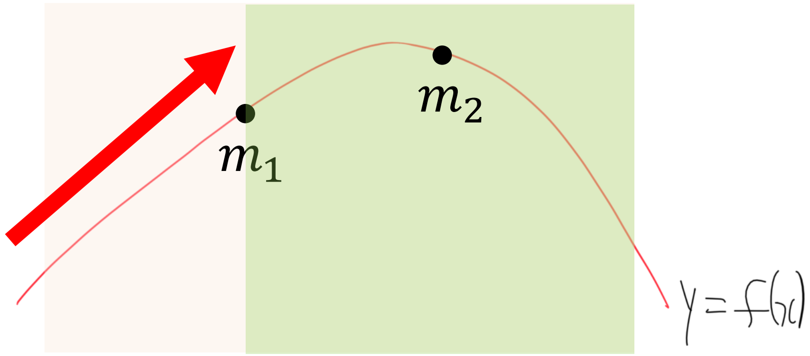




Ternary search (cont.)

- If we set $m_1 = \frac{2l+r}{3}$ and $m_2 = \frac{l+2r}{3}$, in either case the length of the interval becomes $\frac{2}{3}$ of the original.
- If we repeat these steps s times, the length of the interval becomes $(r-l)\times\left(\frac{2}{3}\right)^s$. Try to fix s to get the desired precision.





Ternary search (cont.)

```
// Precondition: l < a < r
for(int steps = 0; steps < 70; steps++) {
    double m1 = (2 * l + r) / 3.0;
    double m2 = (l + 2 * r) / 3.0;
    if(f(m1) > f(m2)) {
        r = m2;
    }else {
        l = m1;
    }
}
```