How to Program a Calculator

A look at the geometric cosine method

Jake Darby

Project Overview

Introduction

Derivation

Errors

Conclusion

Project Overview

- Numerical Analysis of Common Functions

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots
  - Trigonometric Functions

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots
  - Trigonometric Functions
  - Logarithms and Exponentials

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots
  - Trigonometric Functions
  - Logarithms and Exponentials
- These functions can be complex to approximate well

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots
  - Trigonometric Functions
  - Logarithms and Exponentials
- These functions can be complex to approximate well
- Implemented by modern computers and calculators

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots
  - Trigonometric Functions
  - Logarithms and Exponentials
- These functions can be complex to approximate well
- Implemented by modern computers and calculators
  - Implemented in C

Project Overview

- Numerical Analysis of Common Functions
  - Appoximations
  - Errors
  - Implementation
- Functions examined have been studied since antiquity
  - Square Roots
  - Trigonometric Functions
  - Logarithms and Exponentials
- These functions can be complex to approximate well
- Implemented by modern computers and calculators
  - Implemented in C
  - GNU & MPFR libraries

Trigonometric Functions

Trigonometric Functions

- Looked at sin, cos and tan

Trigonometric Functions

- Looked at sin, cos and tan
  - Studied as far back as the Egyptians

Trigonometric Functions

- Looked at sin, cos and tan
  - Studied as far back as the Egyptians
  - Use Trigonometric Identities to reduce the problem

Trigonometric Functions

- Looked at sin, cos and tan
  - Studied as far back as the Egyptians
  - Use Trigonometric Identities to reduce the problem
- Analysed several different methods

Trigonometric Functions

- Looked at sin, cos and tan
  - Studied as far back as the Egyptians
  - Use Trigonometric Identities to reduce the problem
- Analysed several different methods
  - Each of the methods have their benefits

Trigonometric Functions

- Looked at sin, cos and tan
  - Studied as far back as the Egyptians
  - Use Trigonometric Identities to reduce the problem
- Analysed several different methods
  - Each of the methods have their benefits
  - Taylor

Trigonometric Functions

- Looked at sin, cos and tan
    - Studied as far back as the Egyptians
    - Use Trigonometric Identities to reduce the problem
- Analysed several different methods
    - Each of the methods have their benefits
    - Taylor
    - CORDIC

Trigonometric Functions

- Looked at sin, cos and tan
  - Studied as far back as the Egyptians
  - Use Trigonometric Identities to reduce the problem
- Analysed several different methods
  - Each of the methods have their benefits
  - Taylor
  - CORDIC
  - Geometric

Geometric Method Figure 1

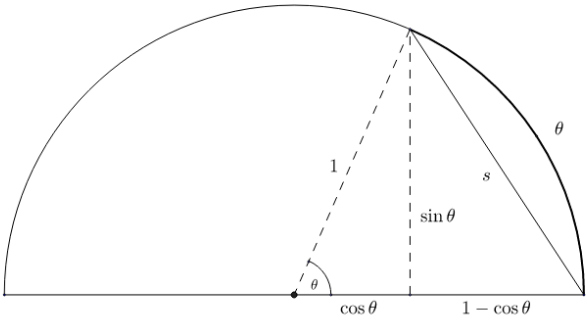
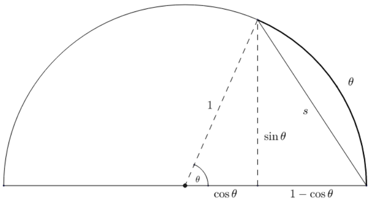


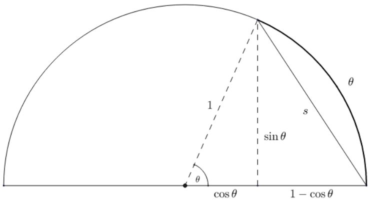
Figure: First diagram in developing the geometric method

Geometric Method Derivation 1



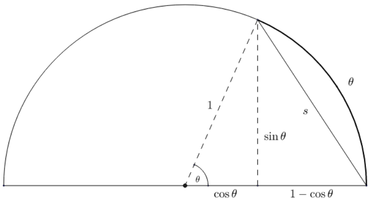
Geometric Method Derivation 1

- $s^2 = \sin^2\theta + (1 - \cos\theta)^2$



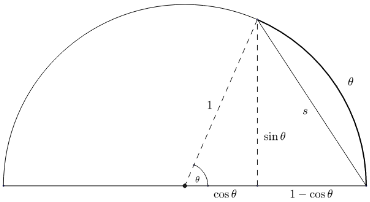
Geometric Method Derivation 1

- $s^2 = \sin^2\theta + (1 - \cos\theta)^2$
- $s^2 = 2 - 2\cos\theta$



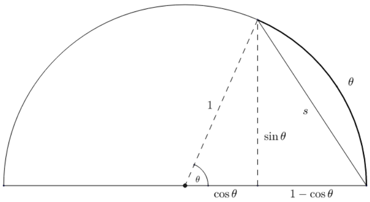
Geometric Method Derivation 1

- $s^2 = \sin^2\theta + (1 - \cos\theta)^2$
- $s^2 = 2 - 2\cos\theta$
- $\cos\theta = 1 - \frac{1}{2}s^2$



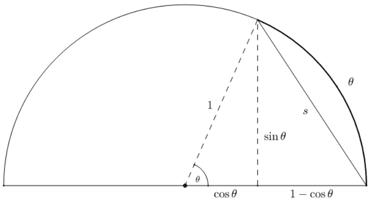
Geometric Method Derivation 1

- $s^2 = \sin^2\theta + (1 - \cos\theta)^2$
- $s^2 = 2 - 2\cos\theta$
- $\cos\theta = 1 - \frac{1}{2}s^2$
- $s \approx \theta$



Geometric Method Derivation 1

- $s^2 = \sin^2\theta + (1 - \cos\theta)^2$
- $s^2 = 2 - 2\cos\theta$
- $\cos\theta = 1 - \frac{1}{2}s^2$
- $s \approx \theta$
- We wish to improve our approximation of s



Geometric Method Diagrams 2

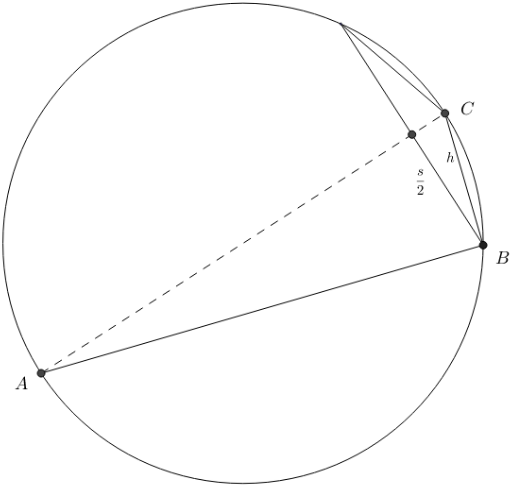
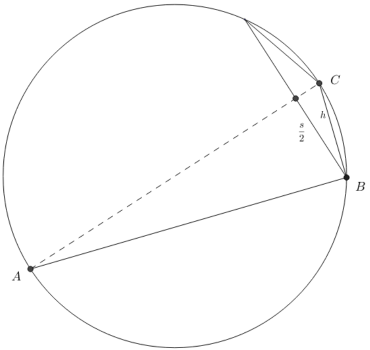


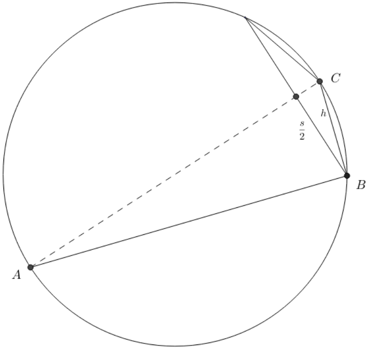
Figure: Diagram to show how to recursively calculate s

Geometric Method Derivation 2



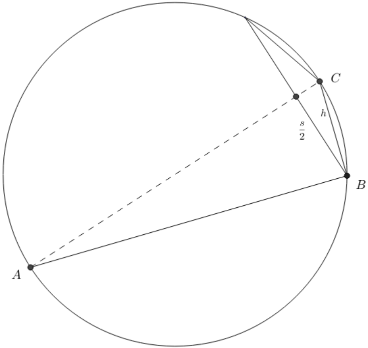
Geometric Method Derivation 2

- ABC is right angled



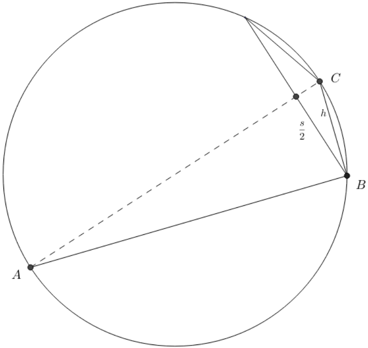
Geometric Method Derivation 2

- ABC is right angled
- $AB = \sqrt{AC^2 - BC^2}$
  $= \sqrt{4 - h^2}$



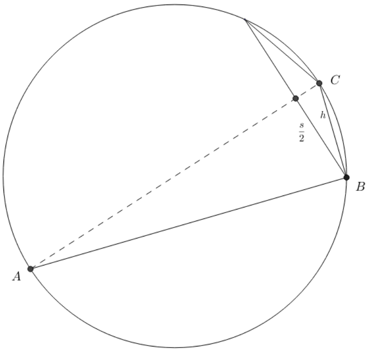
Geometric Method Derivation 2

- ABC is right angled
- $$\begin{aligned} AB &= \sqrt{AC^2 - BC^2} \\ &= \sqrt{4 - h^2} \end{aligned}$$
- Area of ABC:



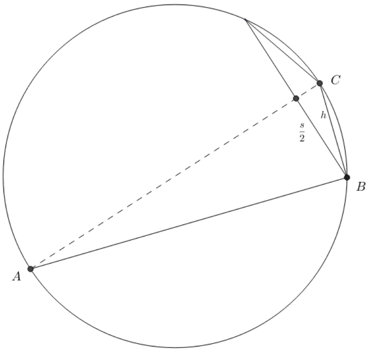
Geometric Method Derivation 2

- ABC is right angled
- $AB = \sqrt{AC^2 - BC^2}$
  $= \sqrt{4 - h^2}$
- Area of ABC:
  - $\frac{1}{2}h\sqrt{4 - h^2}$



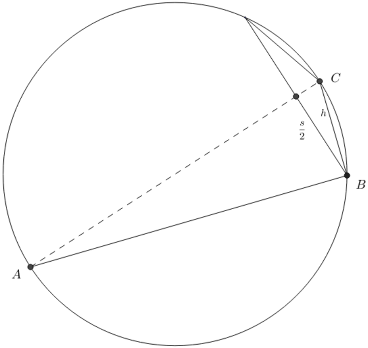
Geometric Method Derivation 2

- ABC is right angled
- $AB = \sqrt{AC^2 - BC^2}$
  $= \sqrt{4 - h^2}$
- Area of ABC:
  - $\frac{1}{2}h\sqrt{4 - h^2}$
  - $\frac{1}{2} \cdot 2 \cdot \frac{s}{2}$



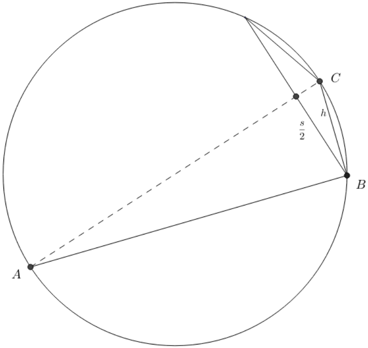
Geometric Method Derivation 2

- ABC is right angled
- $AB = \sqrt{AC^2 - BC^2}$
  $= \sqrt{4 - h^2}$
- Area of ABC:
  - $\frac{1}{2}h\sqrt{4 - h^2}$
  - $\frac{1}{2} \cdot 2 \cdot \frac{s}{2}$
- $s^2 = h^2(4 - h^2)$



Geometric Method Derivation 2

- ABC is right angled
- $AB = \sqrt{AC^2 - BC^2}$
  $= \sqrt{4 - h^2}$
- Area of ABC:
  - $\frac{1}{2}h\sqrt{4 - h^2}$
  - $\frac{1}{2} \cdot 2 \cdot \frac{s}{2}$
- $s^2 = h^2(4 - h^2)$
- $h \approx \frac{\theta}{2}$



Geometric Method Algorithm


```
geometric_cos(θ ∈ [0, π/2], k ∈ ℕ):
    h_0 := θ2^(-k)
    n := 0
    while n < k:
        h_{n+1}^2 := h_n^2 · (4 − h_n^2)
        n ↦ n + 1
    return 1 − (1/2) h_k^2
```

$$\text{geometric\_cos}(\theta \in [0, \tfrac{\pi}{2}], k \in \mathbb{N}):$$
$$h_0 := \theta 2^{-k}$$
$$n := 0$$
$$\text{while } n < k:$$
$$h_{n+1}^2 := h_n^2 \cdot (4 - h_n^2)$$
$$n \mapsto n + 1$$
$$\text{return } 1 - \tfrac{1}{2} h_k^2$$

Introduction
Derivation
Errors
Conclusion

Geometric Method Implementation

```c
#include <assert.h>
#include <math.h>

double geometric_cos(double theta,
                     unsigned int k){
    //k > 0
    assert(k);
    //0 <= theta < pi/2
    assert(0 <= theta < 1.57079632679);

    double h = theta * pow(2, -k);
    h *= h;

    for(int i = 0; i < k; ++i)
        h = h * (4 - h);

    return 1 - h/2;
}
```

Basics and Assumptions

Basics and Assumptions

- Errors occur due to the assumption that $h_0 = \theta 2^{-k}$

Basics and Assumptions

- Errors occur due to the assumption that $h_0 = \theta 2^{-k}$
- We are concerned here with the absolute error

Basics and Assumptions

- Errors occur due to the assumption that $h_0 = \theta 2^{-k}$
- We are concerned here with the absolute error
  - If $\bar{x} \approx x$, then the absolute error is:

$$\epsilon_x := |x - \bar{x}|$$

Basics and Assumptions

- Errors occur due to the assumption that $h_0 = \theta 2^{-k}$
- We are concerned here with the absolute error
  - If $\bar{x} \approx x$, then the absolute error is:

$$\epsilon_x := |x - \bar{x}|$$

- Assumptions:

Basics and Assumptions

- Errors occur due to the assumption that $h_0 = \theta 2^{-k}$
- We are concerned here with the absolute error
  - If $\bar{x} \approx x$, then the absolute error is:

$$\epsilon_x := |x - \bar{x}|$$

- Assumptions:
  - All calculations are performed without error

Basics and Assumptions

- Errors occur due to the assumption that $h_0 = \theta 2^{-k}$
- We are concerned here with the absolute error
  - If $\bar{x} \approx x$, then the absolute error is:

$$\epsilon_x := |x - \bar{x}|$$

- Assumptions:
  - All calculations are performed without error
  - All calculations are performed to arbitrary precision

Error Analysis 1

Error Analysis 1

- Two important propositions:

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$
  - 4.3.2: $h_n > 2\sin(\theta 2^{n-k-1})$

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$
  - 4.3.2: $h_n > 2\sin(\theta 2^{n-k-1})$
- Proposition 4.3.3:

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$
  - 4.3.2: $h_n > 2\sin(\theta 2^{n-k-1})$
- Proposition 4.3.3:
  - $\epsilon_n = |h_n - 2\sin(\theta 2^{n-k-1})|$

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$
  - 4.3.2: $h_n > 2\sin(\theta 2^{n-k-1})$
- Proposition 4.3.3:
  - $\epsilon_n = |h_n - 2\sin(\theta 2^{n-k-1})|$
  - $\epsilon_k < 2^k \epsilon_0$

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$
  - 4.3.2: $h_n > 2\sin(\theta 2^{n-k-1})$
- Proposition 4.3.3:
  - $\epsilon_n = |h_n - 2\sin(\theta 2^{n-k-1})|$
  - $\epsilon_k < 2^k \epsilon_0$
  - Proven by showing $\epsilon_{n+1} < 2\epsilon_n$

Error Analysis 1

- Two important propositions:
  - 4.3.1: $h_n = 2\sin(2^n \sin^{-1}(\theta 2^{-k-1}))$
  - 4.3.2: $h_n > 2\sin(\theta 2^{n-k-1})$
- Proposition 4.3.3:
  - $\epsilon_n = |h_n - 2\sin(\theta 2^{n-k-1})|$
  - $\epsilon_k < 2^k \epsilon_0$
  - Proven by showing $\epsilon_{n+1} < 2\epsilon_n$
  - Uses simple trigonometry and algebraic re-arrangement

Error Analysis 2

Error Analysis 2

- $\epsilon_k = h_k - s$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$
- Let $\mathcal{C} := 1 - \frac{1}{2}h_k^2$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$
- Let $\mathcal{C} := 1 - \frac{1}{2}h_k^2$
- Let $\epsilon_{\mathcal{C}} := |\mathcal{C} - \cos\theta|$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$
- Let $\mathcal{C} := 1 - \frac{1}{2}h_k^2$
- Let $\epsilon_{\mathcal{C}} := |\mathcal{C} - \cos\theta|$
- We can show that $\epsilon_{\mathcal{C}} < 2\epsilon_k$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$
- Let $\mathcal{C} := 1 - \frac{1}{2}h_k^2$
- Let $\epsilon_{\mathcal{C}} := |\mathcal{C} - \cos\theta|$
- We can show that $\epsilon_{\mathcal{C}} < 2\epsilon_k$
- Thus $\epsilon_{\mathcal{C}} < 2^{k+1}\epsilon_0$

$$= 2\theta - 2^{k+2}\sin(\theta 2^{-k-1})$$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$
- Let $\mathcal{C} := 1 - \frac{1}{2}h_k^2$
- Let $\epsilon_{\mathcal{C}} := |\mathcal{C} - \cos\theta|$
- We can show that $\epsilon_{\mathcal{C}} < 2\epsilon_k$
- Thus $\epsilon_{\mathcal{C}} < 2^{k+1}\epsilon_0$

  $$= 2\theta - 2^{k+2}\sin(\theta 2^{-k-1})$$
- $2^{k+2}\sin(\theta 2^{-k-1}) = 2\theta - \frac{1}{6}\theta^3 2^{-2k-1} + \frac{1}{120}\theta^5 2^{-4k-1} + \cdots$

Error Analysis 2

- $\epsilon_k = h_k - s$
- $\epsilon_0 = \theta 2^{-k} - 2\sin(\theta 2^{-k-1})$
- Let $\mathcal{C} := 1 - \frac{1}{2}h_k^2$
- Let $\epsilon_{\mathcal{C}} := |\mathcal{C} - \cos\theta|$
- We can show that $\epsilon_{\mathcal{C}} < 2\epsilon_k$
- Thus $\epsilon_{\mathcal{C}} < 2^{k+1}\epsilon_0$
  $$= 2\theta - 2^{k+2}\sin(\theta 2^{-k-1})$$
- $2^{k+2}\sin(\theta 2^{-k-1}) = 2\theta - \frac{1}{6}\theta^3 2^{-2k-1} + \frac{1}{120}\theta^5 2^{-4k-1} + \cdots$
- $\epsilon_{\mathcal{C}} < \frac{1}{6}\theta^3 2^{-2k-1} + \mathcal{O}(2^{-4k-1})$

Digits of Accuracy

Introduction
Derivation
Errors
Conclusion

Digits of Accuracy

- $2\theta - 2^{k+2}\sin(\theta 2^{-k-1}) < 10^{-N}$

Digits of Accuracy

- $2\theta - 2^{k+2}\sin(\theta 2^{-k-1}) < 10^{-N}$
  - Guarantee N digits of accuracy

Digits of Accuracy

- $2\theta - 2^{k+2}\sin(\theta 2^{-k-1}) < 10^{-N}$
  - Guarantee N digits of accuracy
  - Must solve $2^{k+2}\sin(\theta 2^{-k-1}) > 2\theta - 10^{-N}$

Digits of Accuracy

- $2\theta - 2^{k+2}\sin(\theta 2^{-k-1}) < 10^{-N}$
  - Guarantee N digits of accuracy
  - Must solve $2^{k+2}\sin(\theta 2^{-k-1}) > 2\theta - 10^{-N}$
- Use a test value of $\theta = 0.5$

Digits of Accuracy

- $2\theta - 2^{k+2}\sin(\theta 2^{-k-1}) < 10^{-N}$
  - Guarantee N digits of accuracy
  - Must solve $2^{k+2}\sin(\theta 2^{-k-1}) > 2\theta - 10^{-N}$
- Use a test value of $\theta = 0.5$

N	k
5	6
10	14
50	80
100	163
1000	1658

Figure: This table shows the minimum k required to guarantee N digits of accuracy for our approximation of $\cos(0.5)$

Final Remarks

Final Remarks

- Interesting method, but better exist

Final Remarks

- Interesting method, but better exist
  - e.g. Taylor Series method

Final Remarks

- Interesting method, but better exist
  - e.g. Taylor Series method
- Fairly trivial to reverse the algorithm to find $\cos^{-1}$

Final Remarks

- Interesting method, but better exist
  - e.g. Taylor Series method
- Fairly trivial to reverse the algorithm to find $\cos^{-1}$
- Just one method, in one section

Final Remarks

- Interesting method, but better exist
  - e.g. Taylor Series method
- Fairly trivial to reverse the algorithm to find $\cos^{-1}$
- Just one method, in one section
  - Digit by digit square root

Final Remarks

- Interesting method, but better exist
  - e.g. Taylor Series method
- Fairly trivial to reverse the algorithm to find $\cos^{-1}$
- Just one method, in one section
  - Digit by digit square root
  - Hardware implementable trig calculations

Final Remarks

- Interesting method, but better exist
  - e.g. Taylor Series method
- Fairly trivial to reverse the algorithm to find $\cos^{-1}$
- Just one method, in one section
  - Digit by digit square root
  - Hardware implementable trig calculations
  - Continued fractions for exponentials

Thank you for listening

Thank you for listening

Project at:
https://github.com/Ybrad/Year-4-Project

Thank you for listening

Project at:
https://github.com/Ybrad/Year-4-Project

Any questions?