How to Program a Calculator Presentation (To Change)
Bleep

Jake Conkerton-Darby

- Welcome, hopefully quick so we can all get our lunch
- Jake Darby and as can be seen looking at how to program a calculator
- Not got time for everything, so looking in a bit of detail at one method of calculating cos

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

- <u>INTRO:</u>like to give v. brief overview of whole project
- >Project deals with Numerical Analysis of Common Functions
- This analysis includes
- >How the functions are approximated
- Typically create an algorithm i.e. process which will approximate the correct function value
- >The error of the functions e.g. absolute error
- Examine how to calculate or approximate these errors
- >The implementation of the algorithm
- Includes computational complexity of algorithm, often can be improved with analysis
- >The functions I examined in my project have typically been studied since antiquity
- In particular I studied >Square Root functions, >Trig functions and >Logarithmic and exponential functions
- >The common feature of these functions is that they can all be complex to approximate well
- This has been a problem as these functions have many useful practical applications.
- >These functions are implemented in modern computers and calculators
- The aim of this project is to understand how they are implemented
- >I implemented in C for speed
- >used Gnu Multiple Precision (GNU) and Gnu MPFR (MPFR)
- Allows arbitrary precision in calculations
- Allowed me to calculate $\sqrt{2}$ to 1 million places in about 1 sec

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

- <u>INTRO:</u>The section we're looking at is the Trigonometric Fnctns
- >In particular this section deals with sin, cos and tan
- Many uses over the years, particularly in architecture
- >As such studied since Ancient Egypt
- Used in the building of the pyramids
- Possibly used further back by the Babylonians, but that is debated
- >Typically use trig idents to reduce the problem
- If we find sin or cos, we can find all the others if we know π
- >In the course of the project I analysed several diffent methods
- >Each method works in a slightly different way with different porperties, which gives different benefits
- >The Taylor method uses a series to approximate the values
- These are easy to calculate and analyse
- >The Coordinate Rotation Digital Computer (aka CORDIC) uses rotations of unit vectors via Matrix Transformations
- Has applications in very basic compute systems
- Can also be implemented in hardware
- >The final method is the one we'll be looking at in more detail here

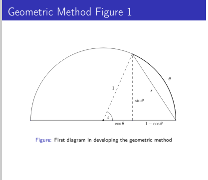
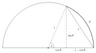


Figure: First diagram in developing the geometric method

- I will be briefly demonstrating how the geometric method is constructed in the following slides.
- This is a sectn of a circle with radius 1
- Detailed are the useful values that we'll be using on this diagram, which we'll go over in detail on the next slide.
- note that only consider $\theta \in [0, \frac{\pi}{2}$ as other values can be reduced to this range via trig idents

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



- <u>INTRO:</u>In analysing this we start by looking at theright hand trinagle
- >This gives the this equation by using Pythagarus' Theorem
- By using the basic trig identity of s2 + c2 = 1 we get>
- Finally re-arranging we get >this which gives us a formula for cos if we know s
- As r = 1, then we know the arc of the angle is θ in length
- The chord approximates this so > $s \approx \theta$
- >The approximation is fairly poor, so we need to improve our approximation of s

Geometric Method Diagrams 2

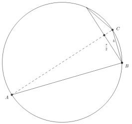
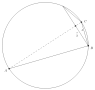


Figure: Diagram to show how to recursively calculate s

- This diagram will allow us to find a much better approximation for s, as detailed on the next slide
- In particular we note that h is the chord of half the original arc

Geometric Method Derivation 2

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



- Now in considering this diagram, first observation >ABC Right Angled by elementary geometry
- Then by Pythagarus we get >
- >To tie it all together we consider the area of ABC
- By using BC as the base of the triangle we get >
- Also by using AC as the base of the triangle we get >
- >We get this by equating the two sides and simplifying
- Thus if we know h accurately we can calculate s^2
- $>h \approx \frac{\theta}{2}$
- repeating the process would give a a new $h \approx \frac{\theta}{4}$, then $\frac{\theta}{8}$, etc...

Geometric Method Algorithm

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

- Thus we get this algorithm
- starts with the approximation that $h_0 := \theta 2^{-k}$
- Uses the recursive formula, that I mentioned on the previous slide
- Finally $h_k = s$

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;
}
```

- Simple implementation in C
- checks bounds with assert commands
- only uses one h variable which is updated

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

- There is only one real source of error
- >Lies in our assumption $h_{\mathbf{0}} = \theta 2^{-k}$
- Actually $h_{\mathbf{0}}$ is close, but not equal.
- Error measure we will be considering is >absolute error
- >That is that if $\bar{x}$ approximates x
- Then the absolute error is ϵ_x
- >All of what follows relies on the following assumptions
- >All calculations are performed correctly
- That is no errors are ever made in the calculations
- >All the calculation are performed to arbitrary precision
- That is we can be working accurately with any number of decimal places
- This is not the case in most computer systems, which only have a finite precision

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

- >When analysing the error of the method, I proved two propositions
- >The first is proposition 4.3.1
- This states that h_n is the length of an arc with angle related to θ.
- Used as a lemma in other proofs
- Simple to prove via induction
- >The second is that $h_n > 2\sin(\theta 2^{n-k-\mathbf{1}})$
- This is proven using the taylor expansion of sin
- This allows us to conclude that h_n is always an overestimate
- This result allows us to then prove >propsition 4.3.3
- >If we define ϵ_n as shown on the screen
- >Then $\epsilon_k < 2^k \epsilon_{\mathbf{0}}$
- >To prove this I showed that $\epsilon_{n+\mathbf{1}} < 2\epsilon_n$
- >Which was done via simple trig and re-arangement
- Now we would like to know if this is a useful error measure
- i.e. the error goes to 0 as k goes to infinity

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 $\epsilon_{\mathcal{C}} < 2^{k+1}\epsilon_0$
  $= 2\theta - 2^{k+2}\sin(\theta 2^{-k-1})$
- $2^{k+2}\sin(\theta 2^{-k-1}) = 2\theta - \frac{1}{6}\theta^3 2^{-2k-1} + \frac{1}{120}\theta^5 2^{-4k-1} + \cdots$
- $\epsilon_{\mathcal{C}} < \frac{1}{6}\theta^3 2^{-2k-1} + \mathcal{O}(2^{-4k-1})$

- We will surmize that > $\epsilon_k = h_k - s$ and > $\epsilon_{\mathbf{0}} = \theta 2^{-k} - 2\sin(\theta 2^{-k-\mathbf{1}})$
- Also we consider >script C to be the approximation of cos
- >and thusly ϵ script C to be the absolute error of the algorithm
- It fairly easy to show that $\epsilon_{\mathcal{C}} < 2\epsilon_k$
- >Therefore we get this final inequality for $\epsilon_{\mathcal{C}}$
- the equality is from substituting in $\epsilon_{\mathbf{0}}$
- It is still not obvious that limit of ep C = 0 as k to infty
- By using the taylor expansion of sin again we get >this
- Substituing this into the previous we get >this equation
- as this is $\mathcal{O}(2^{-\mathbf{2}k-\mathbf{1}})$ then it is obvious the RHS goes to 0 as k goes to infty
- Hence lim ep C = 0 as k to infty, and we conclude the method correctly converges

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)

- >If we have k which validates this inequality then we know that ep C is also less than 10^{-N}
- >This will guarantee us N decimal places of accuracy
- The actual accuracy may be more than this
- >Equivalent to finding k s.t. this eqn is satisfied
- >Using a test value of 0.5, I found solutions to this
- >This table details the minium k to get N digits of accuracy
- We can see that the growth of k required is roughly linear
- We could use this data to calculate a sufficient k, given a required N digits
- For example if we are told we need 10,000 digits we could use k = 18,000 to be sufficient.

2016-06-02

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

- >While this method interests me and I wished to share it with you better methods do exist for approximating cos
- >For example the Taylor Series Method works better
- However it is relatively uninteresting for a presentation
- >One observation is that if you consider the algorithm presented, it is fairly easy to reverese
- Thus if you start with a value for $\cos\theta$ you can approximate $\cos^{-1}\theta$
- >This method presented here is just one of many in the project
- Some other interesting methods I discuss include
- >A square root method that gives precisely the next digit of the root each time in sequence
- >A method of calculating trig values that can be implemented in circuitary hardware
- Uses the CORDIC algorithm, which isn't as accurate, but is very fast
- >A method of using continued fractions to approximate natural exponentials

BLAH

Thank you for listening

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

Any questions?

- I hope this presentation was interesting for some of you
- >If you wish to view my project in it's entirety you can visit this github repository
- This contains all the latex files, pdfs, images and code used
- >Now any questions?