CSC338 Winter 2022

Week 1 - Welcome

Ilir Dema

University of Toronto

Jan 14, 2022

WELCOME TO CSC338

WHAT IS SCIENTIFIC COMPUTING?

Numerical Methods a.k.a Scientific Computing is design and analysis of algorithms for numerically solving mathematical problems in science and engineering

WHERE DO NUMERIC METHODS COME FROM?

AREA OF THE CIRCLE IN ANCIENT EGYPT: $(\frac{8d}{9})^2$

Below there is a problem from Papyrus of Ahmes, around 1550 BC.

A circular field has diameter 9 khet. What is its area?

(A ***khet*** is a length measurement of about 50 meters.)

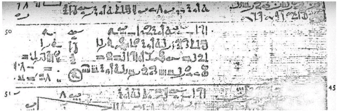
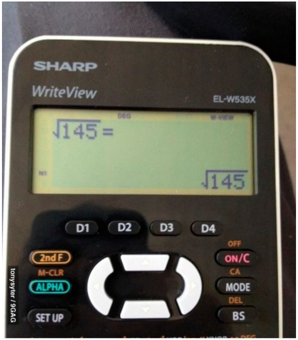


Photo of Problem 50

Solution:

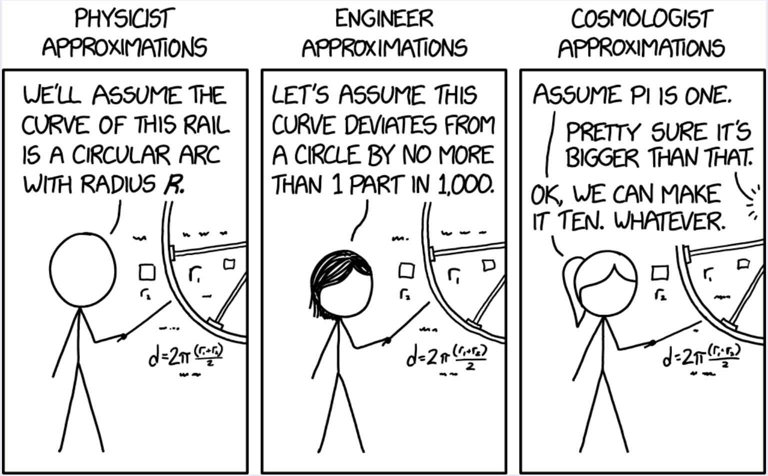
Take away thou
1/9 of it, namely 1;
the remainder is 8.
Multiply it by 8; becomes it 64;
the amount of it,
this is area, 64 setat.

WHY DO WE NEED APPROXIMATIONS?



We deal with continuous quantities

Considering effects of approximations



https://xkcd.com/2205/

CSC338 Topics

- What is approximation, kinds of errors, accuracy (this week)
- Representing continuous quantities using discrete hardware
  - Floating Point Numbers
- Systems of linear equations
- Linear Least Squares
- Non-linear equations
- Non-linear optimization
- PCA (Principal Component Analysis)

Contact Info

ilir.dema@utoronto.ca

Lecture	Office Hours
Fridays 11am-1pm	Tuesdays 3pm-4pm

Course website: http://mcs.utm.utoronto.ca/~338

LECTURES & TUTORIALS

- Lectures
  - As long as we are online, lectures will take place on zoom `https://utoronto.zoom.us/j/83347392030`
  - Also you will complete a small interactive quiz to earn lecture participation mark
- Tutorials
  - Held every week, starting Jan 19
  - They will be dedicated to problem solving
  - You will complete a simple interactive quiz to earn participation mark

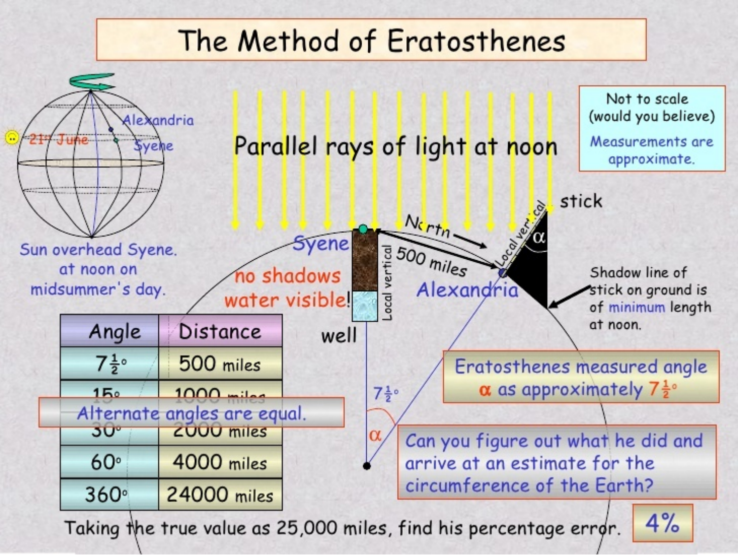
Course Structure

Work	Weight	Comment
Assignments	30%	three of them , with equal value
Tutorials	5%	need to participate in 8 of them
Lecture	5%	need to participate in 8 of them
Term Test	20%	Mar 4, 2022
Final Exam	35%	TBA
Floating	5%	Aded to your best (MT or Final)

Academic Integrity

- Please do not cheat because:
    - You're not helping yourself
    - You WILL get caught
- How to avoid plagiarism
    - Everything you submit should be done by you
    - Never look at another person's work
    - Never show anyone your work
    - This goes for drafts, partial solutions, etc

Scientific Computations



https://www.quora.com/profile/Christopher-Ducey

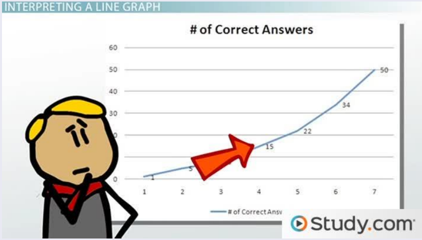
What are Eratosthenes sources of error?

Hint: The answer is not "Earth is flat"

- Earth is not a perfect sphere
- Distance from Syene to Alexandria has been measured empirically
- The value for the angle α has been truncated
- The value for π has been truncated as well

Steps for solving a computational problem

- Develop a mathematical model. That often involves:
  - Replace infinite with finite, continuous with discrete
  - Differential with algebraic
  - Nonlinear with linear
- Develop an algorithm to solve the problem numerically. **This is the focus of this course.**
- Run the algorithm and interpret the results (we won't do that)



Well Posed Problems

- Problem is well-posed if solution
  - exists
  - is unique
  - depends continuously on problem data
- Otherwise, problem is ill-posed
- **In this course, we will focus on well-posed problems.**

Well-Posed or Ill-Posed?

- Compute $f(x) = x^2 - 1$
- Solve $x^2 - 1 = 0$
- Solve $x^3 - x^2 + x - 1 = 0$
- Solve $2x - 1 = 0$ in integers
- Transform a colour image to gray scale
- Transform a gray scale image to colour
- Map numerical grade to letter grade

Sensitivity

Even if problem is well posed, solution may still be sensitive to input data

Example - compute $\tan(x)$

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [1]:

```python
import math
x = math.pi/2-0.0001
y = math.pi/2-0.0002
print(math.tan(x), math.tan(y))
```

```
9999.999966661644 4999.999933332353
```

Sensitivity

Computational algorithm should not make sensitivity worse

Example - quadratic formula

In [10]:
```
from math import sqrt
from decimal import *

getcontext().prec = 4

def solve_quad(a, b, c):
    a, b, c = Decimal(a), Decimal(b), Decimal(c)
    two, four = Decimal(2.0), Decimal(4.0)
    d = Decimal(sqrt(b*b-four*a*c))
    return (-b+d)/two/a, (-b-d)/two/a

a, b, c = 0.05010, -98.78, 5.015
#
# Correct roots are 1971.605916, 0.05077069387
#
r1, r2 = solve_quad(a, b, c)
print("Naively computed roots are: {}, {}".format(r1, r2))
```

```
Naively computed roots are: 1972, 0.07519
```

In [11]:
```
#
# Compare:
#
x1, x2 = 0.05077069387, 0.07519
print(a*x1*x1+b*x1+c, a*x2*x2+b*x2+c)
```

```
4.5553694150157753e-10 -2.4119849578413906
```

What is accuracy of a solution?

Definition

Accuracy is the "closeness" of a computed solution the actual solution.

Q: How to evaluate accuracy of a solution?

- Absolute Error: |approx.value - true value|
- Relative Error: (absolute error)/(true value)

Estimate errors into computing Earth's surface

- Model: $A = 4\pi r^2$, $r = 6371km, \pi = 3.14$.
- True value (known from other sources): $510.1 \times 10^6 km^2$
- Approx. value: $4 \times 3.14 \times 6371^2 \approx 509.8 \times 10^6 km^2$
- AE: $|510.1 \times 10^6 km^2 - 509.8 \times 10^6 km^2| = 0.3 \times 10^6 km^2$
- RE: $\frac{0.3 \times 10^6}{510.1 \times 10^6} \approx 0.06\%$

More errors

Problem: compute $\cos \frac{\pi}{6}$

- True input: $x = \frac{\pi}{6} = 0.5235987755982988$
- Approx.input $\hat{x} = 0.524$
- True function: $f(x) = \cos x$
- Approximate function $\hat{f}(x) = 1 - \frac{x^2}{2}$
  - Obtained by truncating $\cos x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$

Forward and Backward Error

Forward and Backward Error

Suppose we want to compute $y = f(x)$, where $f\colon \mathbb{R} \to \mathbb{R}$, but obtain approximate value $\hat{y}$

Forward error $= \Delta y = \hat{y} - y$

Backward error $= \Delta x = \hat{x} - x$, where $f(\hat{x}) = \hat{y}$

$$
\begin{tikzcd}
x \arrow[r, "f"] \arrow[dr, "\hat{f}"] \arrow[d, leftrightarrow, "\text{backward error}"'] & y = f(x) \arrow[d, leftrightarrow, "\text{forward error}"] \\
\hat{x} \arrow[r, "f"'] & \hat{y} = \hat{f}(x) = f(\hat{x})
\end{tikzcd}
$$

Example

Back to $\cos\frac{\pi}{6}$

- $\hat{f}(x) = 1 - \frac{x^2}{2}, \quad f(x) = cos(x)$
- $\hat{x} = 0.524, \quad x = \frac{\pi}{6}$
- Total error: $\hat{f}(\hat{x}) - f(x)$
  $= \hat{f}(\hat{x}) - f(\hat{x}) + f(\hat{x}) - f(x)$
- Forward error:
  - Computational error: $\hat{f}(\hat{x}) - f(\hat{x})$
  - Propagated data error: $f(\hat{x}) - f(x)$
- Backward error: $\hat{x} - x$

Benefits of backward error

Compute $y = \sqrt{x}$

- Suppose we have a computational procedure that gives us $\hat{y}$, an approximate value for $\sqrt{x}$.
- By definition, absolute error $= |y - \hat{y}| = |\sqrt{x} - \hat{y}|$
- We cannot really compute it as long as we do not know the true value of $\sqrt{x}$.
- But it is easy to evaluate backward error $= |\hat{y}^2 - x|$!
- It is worth saying ancient Babylonians knew this. They would start by an initial guess for the square root of a number s, say x_0, and then improve iteratively: $x_{n+1} = \frac{1}{2}(x_n + \frac{s}{x_n})$

CONDITIONING

DEFINITION

- A problem is *well-conditioned* or *insensitive* if a relative change in the input causes a similar relative change in the solution.
- A problem is *ill-conditioned* or *sensitive* if a relative change in the input causes a much larger relative change in the solution.
- Formally *Conditioning Number* C_N can be defined as follows:
  $C_N = \frac{|\Delta y/y|}{|\Delta x/x|} = \frac{|(f(\hat{x})-f(x))/f(x)|}{|(\hat{x}-x)/x}$
- Otherwise |rel.forward error|=C_N|rel.backward error|
- If $C_N >> 1$ the problem is ill-conditioned.

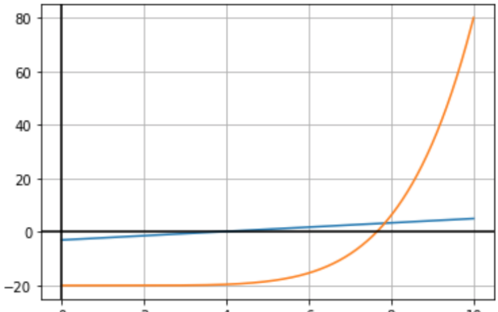
CONDITIONING

ASSUME f IS DIFFERENTIABLE. THEN:

- $C_N = \frac{|f(x+\Delta x)-f(x)|/|f(x)|}{|\Delta x|/|x|} \approx \frac{|xf'(x)|}{f(x)}$
- Please note this definition is not good when $f(x) = 0$ (eg. root finding).
- In such cases, C_N can be defined as the ratio of absolute errors, leading to
- $C_N = 1/|f'(x)|$ (why?) which means the root finding problem is well conditioned if the slope at the root is $>> 1$.

CONDITIONING - ROOT FINDING EXAMPLE

```
import matplotlib.pyplot as plt
x = [i*0.01 for i in range(1001)]
xx = [i*0.001 for i in range(1001)]
y = [0.8*v-3 for v in x]
z = [100*v**6-20 for v in xx]
plt.plot(x, y)
plt.plot(x, z)
plt.grid(True, which='both')
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
plt.show()
```



Conditioning - more examples

$\tan x$

- Computation of $\tan x$ is sensitive for values of x near any multiple of $\pi/2$:

$$C_N = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x(1+\tan^2 x)}{\tan x}\right| = \left|\left(\frac{1}{\tan x} + \tan x\right)\right|$$

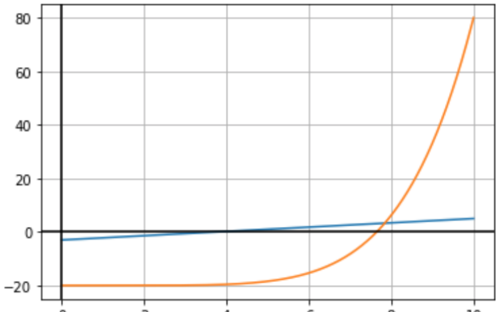
$\sqrt{x}$

- Computation of $\sqrt{x}$ is well conditioned:

$$C_N = \left|\frac{xf'(x)}{f(x)}\right| = \left|\frac{x/(2\sqrt{x})}{\sqrt{x}}\right| = \frac{1}{2}$$

Conditioning

```python
import matplotlib.pyplot as plt
x = [i*0.01 for i in range(1001)]
xx = [i*0.001 for i in range(1001)]
y = [0.8*v-3 for v in x]
z = [100*v**6-20 for v in xx]
plt.plot(x, y)
plt.plot(x, z)
plt.grid(True, which='both')
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
plt.show()
```



Stability and accuracy

Definition

An algorithm is *stable* if the result is relatively insensitive to perturbations during computation.

Note:

An algorithm is *stable* if the result is the exact soltution to a nearby problem. Th is also called "the backward error view".

Q: When can we obtain accurate solutions?

- When the problem is well conditioned, and,
- When the algorithm is stable.

References

Michael T. Heath
Scientific Computing (Revised Second Edition)
SIAM