Numerical Methods - Preliminaries

Y. K. Goh

Universiti Tunku Abdul Rahman

2013

Table of Contents

Outline

Introduction to Numerical Methods

- One of the earliest and most practical branches of mathematics.
- For ancient civilisations, the study of numerical methods, like trigonometry, is extremely important in:
  - constructions, carpentry, taxation, navigation, astronomy, calendar, . . .
- Modern applications:
  - Computational fluid mechanics
  - Rocket trajectories
  - Numerical Weather Prediction
  - Engineering
  - Chemical reactions
  - Betting in football tournament
  - Insurance premiums

Outline

Numerical Methods in Ancient Civilisations

Example (A Babylonian Example)

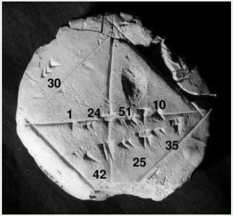
- Put away your calculator, and do this example with only pen and paper.
- Consider a metal of square shape, the lengths on 4 sides are 1 meter, you need to fix a string along two ends of the diagonal, what is the length of string accurate to mm?
- The exact answer is $\sqrt{2}$.
- However to prepare the string we need a numerical value.
- Can you approximate the value of $\sqrt{2}$ accurate to 3 decimals?

Key Concepts:

- exact value, numerical value, approximation, accuracy

Numerical Methods in Ancient Civilisations

Example (A Babylonian Example)



- Babylonian clay tablet – YBC7289 (c. 1800 – 1600 BC). [*Source: Wikipedia*]
- The Babylonian recorded the value of $\sqrt{2}$ in sexagesimal as $1;24,51,10$.
- In our modern language, the value of $\sqrt{2}$ in decimal is $1.4142\ldots$, ie $\sqrt{2} \approx 1 + \frac{24}{60} + \frac{51}{60^2} + \frac{10}{60^3} + \cdots = 1.41421296\ldots$.

Key Concepts:

- number systems: decimal, sexagesimal

Example (How Babylonian calculate $x = \sqrt{S}$)

- First algorithm used for approximating $\sqrt{S}$
- Recorded by the first century Greek mathematician Heron's of Alexandrian.
  - choose a number x_0 as an initial guess for x.
  - improve the guess with $x_1 = \frac{1}{2}\left(x_0 + \frac{S}{x_0}\right)$
  - similarly, we can further improve the approximation by the iterative formula $x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$ for $n = 1, 2, \ldots$.

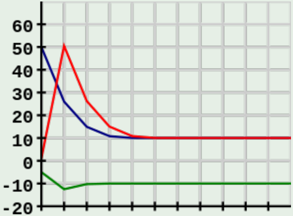


Figure: Convergence of Heron's method to evaluate $\sqrt{100}$.

Key Concepts:

- algorithm, initial value, iteration, convergence, fixed points

Numerical Methods in Ancient Civilisations

Example (How Ancient Indian calculate $x = \sqrt{S}$)

- Recorded in Bakhshali manuscript (approximately 400 AD).
- Algorithm:
  - Let N^2 be the nearest perfect square to S.
  - $d = S - N^2$
  - $P = \dfrac{d}{2N}$, and $A = N + P$
  - $\sqrt{S} \approx A - \dfrac{P^2}{2A}$
- For example, to evaluate $\sqrt{5.3}$, the nearest perfect square is $N^2 = 2^2 = 4$.
- Thus, $d = 1.3, P = 0.325, A = 2.325$, and finally the approximation $\sqrt{5.3} \approx 2.302285$.
- Alternatively, the algorithm can be written as $\sqrt{S} \approx \dfrac{N^4 + 6N^2S + S^2}{4N^3 + 4NS}$.

Key Concepts:

- there are always alternative algorithms to solve the same problem.

Outline

Brief Applications of Various Topics in Numerical Methods

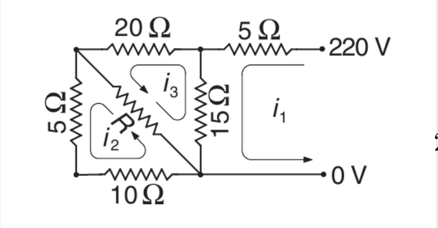
Let breifly outline the topic in numerical methods that we will cover in this course:

- Chapter 2 – Solving non-linear equation(s).
- Chapter 3 – Solving system of linear equations.
- Chapter 3 – Finding eigenvalues
- Chapter 4 – Interpolations
- Chapter 5 – Differentiations and integrations
- Chapter 6 – Ordinary differential equation - initial-value problem
- Chapter 7 – Ordinary differential equation - boundary-value problem
- Chapter 8 – Partial differential equation

In the next few slides, we will see some of the examples how these topics applied in various situations.

Example (Electrical networks)

- The following electrical network can be viewed as consisting of 3 loops.
- Applying Kirchoff's law, $\sum$ voltage drops $= \sum$ voltage sources to each loop, we get:



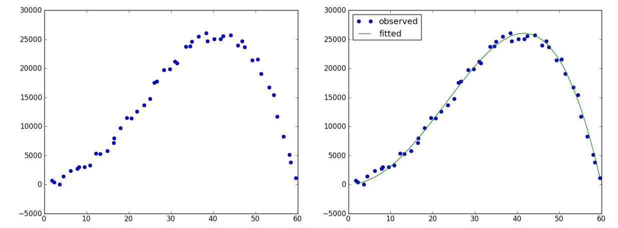
$$5i_1 + 15(i_1 - i_3) = 220$$
$$R(i_2 - i_3) + 5i_2 + 10i_2 = 0$$
$$20i_3 + R(i_3 - i_3) + 15(i_3 - i_1) = 0$$

- What are the value of the currents i_1, i_2 and i_3 if given $R = 4.2$?
- Essentially the system of linear equations can be re-written as a matrix equation and solving for the currents involve inverting a coefficient matrix.

Curve fitting – Finding a function $f(x)$ to describe data

Example (Fitting Data)

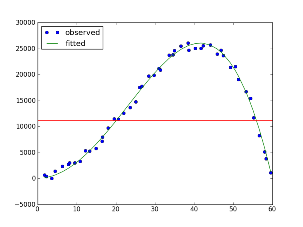
- Consider the observed data (left figure) for the vertical velocity of an UFO.
- The data are recorded poorly, can we find a function $v(t)$ that best fit the velocity data on the UFO?



- We can apply a numerical curve fitting algorithm to the data, and the outcome is a function $v(t) = 40 + 32t^2 - 0.013t^4 + 0.00025t^{4.683}$.
- The fitted function is plotted as a curve in the right figure.

Example (Root finding)

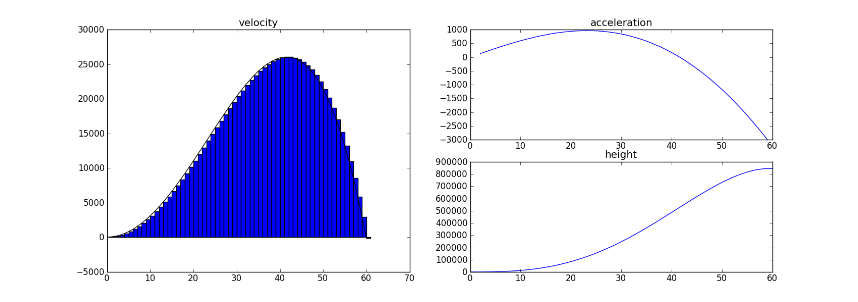
- We know the escape velocity for the Earth is 11200 m/s, find the time at which the UFO reach the Earth escape velocity.



- Mathematically the problem is to find the interceptions for the curve $v(t)$ and straight line $v = 11200$ as in the figure.
- equivalently, we want to find the value of t where $f(t) = 0$, where $f(t) = v(t) - 11200 = 40 + 32x^2 - 0.013x^4 + 0.00025x^{4.683} - 11200.$
- From the bisection method, we found the times that the UFO will reach esscape velocity are $t = 20.14$, or $t = 55.90$.

Example (Root finding)

- Now that we know how the UFO moves vertically, we can further estimate the UFO's height and acceleration.
  - Integration of the vertical velocity gives the height.
  - Differentiation of the vertical velocity gives the acceleration.

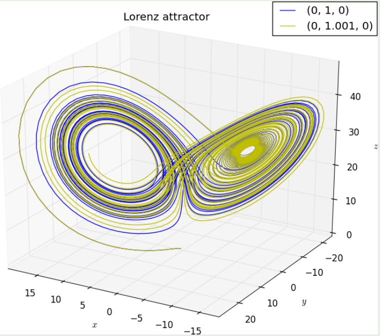


Numerical ODE - Initial Value Problem

Example (Solution of system of ODEs with different initial values)

- In 1963, Edward Lorenz developed a simplified mathematical model for atmospheric convection.

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$



- The simulation on the right uses: $\sigma = 10, \rho = 28$, and $\beta = 8/3$.
- Note that two close by starting points ended up following very different trajectories.

Outline

Numerical methods in practice

- From the previous slides, we know the aim of Numerical Methods is to find a numerical *approximation* to the *exact* solution.
- The approximation should be *close enough* to the exact solution.
  - What is the acceptable *tolerance* for error?
  - Often this tolerance is the *stopping criteria* for the numerical algorithms.
- Since it is an approximated solution, there is always question that how far the approximation from the exact answer:
  - the is to ask: what is the error? (*absolute error*)
  - related questions are: is the error "big"? (*relative error*)
  - what are the possible sources of errors?
- The numerical approximations are the outcomes.
- In Numerical Methods, we are also interested how to get these outcomes:
  - Can we systematically re-produce the outcomes? (*algorithms*)
  - Are there any alternative algorithm that is faster or more accurate?
  - Under what kind of conditions the algorithm will not work? (*ill-posed problems*)
  - Often the algorithm is iterative, will the iterations be stable? (*convergence*)

Sources of Errors

Possible sources of errors:

- Truncation errors
  - "Intrinsic" error in a method
  - often occurs due to representing a series (infinite sum) with a finite sum – e.g. as in Taylor's series.
- Round-off errors
  - Due to limitation of computing device.
  - Computer arithmetic is finite, that is to say computer stores number using a finite amount of memories.
  - Round-off errors arised once an exact decimal number, like 3.28, is stored as a finite memory *floating point* number.
- Data uncertainty (not considered)
- Inappropriate model (not considered)

We will discuss truncation errors and round-off errors in more details.

Outline

Outline

Geometric Series

Example

- To get a feel on what is truncation error, let consider the formula for geometric series:
$$\frac{a}{1-r} = \sum_{n=0}^{\infty} ar^n.$$
- Consider the case where $a = 1$ and $r = \frac{1}{4}$, we have $\displaystyle\frac{4}{3} = \sum_{n=0}^{\infty} 4^{-n}$.
- Here, the RHS is a sum up to infinite many terms of 4^{-n} to give the exact value of the LHS, ie $\frac{4}{3}$.
- However, it is unlikely that a human or a computer can sum up to infinite many terms, we have to stop somewhere.
- Let say we stop after 10 terms, then we have the *approximation* for $\frac{4}{3}$ which is $\sum_{n=0}^{9} 4^{-n} = 1.3333320617675781\ldots$.
- Stopping after 10 steps broke the equal sign and the resulting error is called the truncation error. In this case the truncation error is $\sum_{n=10}^{\infty} 4^{-n}$.

Outline

Approximation by Taylor's series

Let say we would like to calculate $\cos(0.1)$, but do not have a scientific calculator or lookup table.

- We know $\cos(0) = 1$.
- We know all the derivatives of $\cos$ at 0.
- Taylor series for $\cos$ is

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

- Thus, we can "calculate" $\cos(0.1)$ from its Taylor's series.
- Reality: we cannot calculate exactly $\cos(0.1)$, as we are not be able to tabulate all the infinite sum of Taylor's series, but we can *approximate* it with a Taylor's polynomial.
- The series is truncated after a certain term.
- We can generalise the example for geometric series to Taylor series.

Example

- First approximation, $\cos(0.1) \approx 1$
- Subsequent approximation:

$$
\begin{aligned}
\cos(0.1) &\approx 1 - (0.1)^2/2! = 0.9950 \\
\cos(0.1) &\approx 1 - (0.1)^2/2! + (0.1)^4/4! \\
&= 0.99500416666667 \\
\cos(0.1) &\approx 1 - (0.1)^2/2! + (0.1)^4/4! - (0.1)^6/6! \\
&= 0.99500416527778 \\
&\vdots
\end{aligned}
$$

- Numerical approximation correct to 14 d.s.p $\cos(0.1) = 0.99500416527803$.

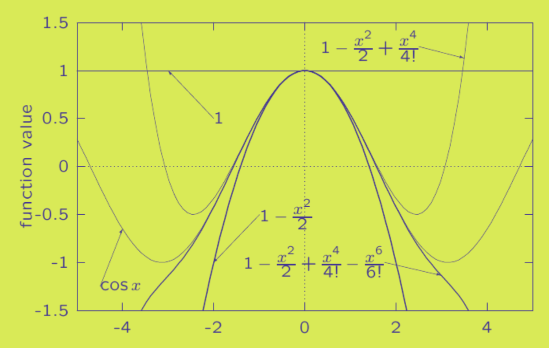


Figure: Taylor's series approximation of $\cos(x)$

Taylor Polynomial and Series

Theorem (Taylor's Theorem)

Suppose $f \in C^n[a,b]$ *and* $f^{(n+1)}$ *exists on* $[a,b]$*. Let* $x_0 \in [a,b]$*. For every* $x \in [a,b]$*, there exists a number* $\xi(x)$ *between* x_0 *and* x *such that*

$$f(x) = P_n(x) + R_n(x)$$

where

$$\begin{aligned} n\text{-th Taylor Polynomial, } P_n(x) &= \sum_{k=0}^{n} \frac{f^k(x_0)}{k!}(x-x_0)^k, \\ \text{Remainder, } R_n(x) &= \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)^{n+1}. \end{aligned}$$

Key Numerical Methods questions

Some of the key questions in numerical analysis:

- What is the estimated value and its estimated error?
- What is the maximum error? (bounds of remainder)
- Let the acceptable error be ϵ_a, how many approximation steps I need to do?
- In what situation the method will not work? (radius of convergence)

Example

Let consider the Taylor's series approximation for $\ln(0.9)$.

- Write down the Taylor's polynomial of order n, $P_n(x)$ and its remainder $R_n(x)$ for $\ln(1+x)$.
- Approximate $\ln(0.9)$ by $P_5(x)$. What is the absolute error?
- What is the maximum error if it is approximated by $P_n(x)$.
- How many terms need to be included, i.e. what is n, in order for the Taylor's polynomial to have error less than an acceptable tolerance of $\epsilon = 10^{-8}$?
- Can we use the Taylor's polynomial to approximate $\ln(1.8)$? Why?
- How about to approximate $\ln(2.4)$? Why?

Outline

Outline

Finite Precision Computer Arithmetic

- Often we use a computer to implement our numerical algorithms.
- We must aware that computer is a finite precision arithmetic machine.
- That is to say, computer uses limited memory to store numbers, and cannot distinguish numbers that is smaller than a fixed value (machine epsilon) from zero.
- Another word, we cannot hope to use a computer to calculate a numerical approximation correct to the number of digits that beyond the resolution of machine epsilon.
- For example, the machine epsilon of a 64-bit floating-point is $\approx 2 \times 10^{-16}$, thus it is not possible (usually) to compute a numerical approximation accurate to 17 decimal places.
- As a result, some real numbers like $1/3 = 0.333333\dots$ will not be represented by computer exactly, but to the best possible within the allocated memory.
- The resulting error is called the round-off error, and the number stored in the finite allocated memory of computer is called the floating-point number.

Example

- The python code below demonstrates that the computer arithmetic is finite:

```
eps = 1.0
while 1.0 - eps < 1.0:
    print "eps = ", eps
    eps = eps/10.
print "Escaped!"
```

- The output looks as follow:

```
eps = 1.0
eps = 0.1
eps = 0.01
...
eps = 1e-15
eps = 1e-16
Escaped!
```

Outline

Computer Number System

Some of the basic "Numbers" in computer:

- Booleans (true or false)
- Integers (unsigned and signed)
- Floating-points (single, double)
- Complex floating-points (`a + jb`)

Computer Integers

There are several schemes to represent numbers in computer. All the schemes are in Base-2.

Example (unsigned 8-bit integers)

- `uint8` or `uchar` numbers are positive integers that stored in 8-bit memory. It can only takes $2^8 = 256$ different values.
- Smallest value of `uint8`: $0000\ 0000_2 = 0$.
- Largest value of `uint8`: $1111\ 1111_2 = 255 = 1 - 2^8$.

Example (signed 8-bit integers)

- `int8` or `char` numbers are `uint8` integers that biased/shifted by $shift = -2^7$.
- Smallest value of `int8`: $0000\ 0000_2 + shift = 0 - 128 = -128$.
- Largest value of `int8`: $1111\ 1111_2 + shift = 127$.

Computer Integers

Example (32-bit integers)

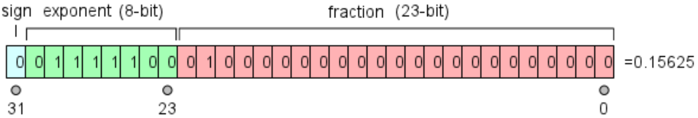
- `int32` are integers that stored in 32-bit memory and baised/shifted by -2^{31}. It can takes $2^{32} = 4294967296$ different values.
- Smallest value of `int32`: $0 - 2^{31} = -2147483648$.
- Largest value of `int32`: $(2^{32} - 1) - 2^{31} = 2147483647$.

Floating-Points

- Real numbers are represented as finite memory floating-point numbers. Usually 32-bit (single precission) or 64-bit (double precission)
- A commonly used floating-point standard is the IEEE 754 standard.
- Any real number may be expressed as $r = sign \times m \times 2^e$, where $m = (0.b_1 b_2 b_3 \ldots)_2$ is called the mantissa and e is the exponent.
- The floating point representation of the real number r is denoted as $fl(r)$, where usually m and e are stored in finite bits of memory.

IEEE 754 Single-precision

- A floating-point format uses 32-bit memory.



- $fl(r) = \begin{cases} (-1)^s \times 2^{e-127} \times (1.f) & \text{normalised, } 1 \leq e \leq 254, \\ (-1)^s \times 2^{e-126} \times (0.f) & \text{denormalised, } e = 0, f > 0, \\ 0 & e = 0, f = 0, \\ (-1)^s Inf, & e = 255, f = 0, \\ NaN, & e = 255, f > 0. \end{cases}$
- s - 1-bit sign bit
- e - 8-bit biased exponent
- m - 23-bit fraction

IEEE 754: Example

Example

What is the value of the IEEE single-precision floating point number represented by the following binaries?

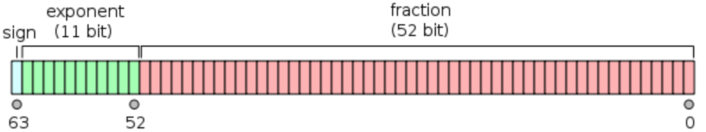
- $00111110\ 00100000\ 00000000\ 00000000$ [Ans.: 0.15625]
- $11111111\ 11111111\ 1111111\ 11111111$ [Ans.: NaN]

Example

- Show that 1313.3125_{10} is 10100100001.0101 in base-2.
- Put the base-2 value into a normalized form.
- Write down the IEEE 754 single-precision representation for 1313.3125_{10}. [Ans.: $1|10001001|01001000010101000000000$]

IEEE 754 Double-precision

- A double floating-point uses 64-bit memory.



- $$fl(r) = \begin{cases} (-1)^s \times 2^{e-1023} \times (1.f) & \text{normalised, } 1 \le e \le 2046, \\ (-1)^s \times 2^{e-1022} \times (0.f) & \text{denormalised, } e = 0, f > 0, \\ 0 & e = 0, f = 0, \\ (-1)^s Inf, & e = 2047, f = 0, \\ NaN, & e = 2047, f > 0. \end{cases}$$
- s - 1-bit sign bit
- e - 11-bit biased exponent
- m - 52-bit fraction

Summary of Range and Precision

Type	Exp. bias	Low Limit	High Limit	Fraction	Precision	Significant digits
Single ($32/23$ fp)	127	$2^{-126} \approx 10^{-38}$	$2^{128} \approx 10^{38}$	23	$2^{-24} \approx 10^{-7}$	7
Double ($64/52$ fp)	1023	$2^{-1022} \approx 10^{-308}$	$2^{1024} \approx 10^{308}$	52	$2^{-53} \approx 10^{-16}$	16

Machine Espilon and Precision

- Range - gives the limits of the values could be represented by a floating-point format. That is the numbers covered by Low Limit and High Limit in the previous slide.
- Machine Epsilon, $\epsilon = 2^{-23}$ (32/23 fp) or 2^{-52} (64/52 fp) is the *smallest* positive machine number such that $1 + \epsilon \neq 1$. Try this: `eps = 2.0**(-52)` `1 + eps` (This give you 1.0000000000000002) `1 + eps/2` (This give you 1.0)
- Errors that can happen when assigning a real number x to a variable in a program:
  - **Overflow:** example: $x = 2^{1920}$, this will maps to the machine number `INF`.
  - **Underflow:** example: $x = 2^{-1920}$, this will maps to the machine number 0.0.
  - **Roundoff error:** This happen when the real number cannot be represented exactly in Base-2, and will be replaced by the nearest machine number.
- In additions, errors or rather unexpected results will happens after certain arithmetic operations. We call this **Loss of significant digits**.

Example

Let $fl(x)$ be the floating-point representation of a real number $x = 1/3$ on a 16-bit register with 1 sign bit, 5 exponent bits with 15 bias, and 10 mantissa bits.

- What is the the machine epsilon?
- What is bit representation of $fl(x)$? [ANS: $fl(x) = 0|01101|0101010101 = 0.3332519531$]
- What is the roundoff error?

- Whenever representing a real number x to a floating-point format $fl(x)$, the conversion suffer from round-off error.
- Let say $2^e \leq x \leq 2^{e+1}$ and the number of bits in fraction field is n, then x will be rounded to its nearest floating-point, which is either
  - $(-1)^s \times (1.a_1a_2\ldots a_n)_2 \times 2^e$, or
  - $(-1)^s \times [(1.a_1a_2\ldots a_n)_2 + \frac{1}{2^n}] \times 2^e$, or
- This gives the rounding error, $\epsilon_{fl} = |x - fl(x)| \leq \frac{1}{2}2^{e-n}$.
- i.e. the relative error,

$$\epsilon = \frac{|x - fl(x)|}{|x|} \leq \frac{\frac{1}{2}2^{e-n}}{(1.a_1a_2\ldots a_n)_2 \times 2^e} \leq \frac{\frac{1}{2}2^{e-n}}{2^e} = 2^{-n-1}.$$

Outline

Loss of Significant Digits – addition

Example

Most computer performing arithmetic operations from left to right, ie. whenever $a + b + c$ it means $(a + b) + c$. Consider addition in a single precision (roughly 7 significant digits) and single register system

- Calculating $10,000,000 + \underbrace{1. + 1. + 1.}_{\text{10 million times}}$, will get $10,000,000$.
- Because $0.1000000 \times 10^8 + 0.1000000 \times 10^1$ gives 0.10000000×10^8.
- On the other hand, $\underbrace{1. + 1. + 1.}_{\text{10 million times}} + 10,000,000 = 20,000,000$ gives the expected answer.

$\boxed{\text{Add numbers in size order to avoid LSD in addition.}}$

Loss of Significant Digits – substraction

Example

- Let's calculate $x - \sin x$ for $x = 1/15$.

$$\begin{aligned} x &= 0.66666\ 66667 \times 10^{-1} \\ \sin x &= 0.66617\ 29492 \times 10^{-1} \\ x - \sin x &= 0.00049\ 37175 \times 10^{-1} \\ &= 0.49371\ 75000 \times 10^{-4} \end{aligned}$$

- 3 significant digits from the right are lost.
- Use Taylor's series for $x - \sin x \approx x^3/3! - x^5/5! + x^7/7!$ instead, which gives $0.49371\ 74328 \times 10^{-4}$, and the actual value is $0.49371\ 74327 \times 10^{-4}$.

Substractions of close numbers cause LSD.

Outline

Outline

Definition (Limit)

A function $f(x)$ is said to have a limit L at a if given any number $\epsilon > 0$, we can find an $\delta > 0$ such that $|f(x) - L| < \epsilon$ whenever $0 < |x - a| < \delta$. We write $\lim_{x \to a} f(x) = L$.

Definition (Continuity)

$f(x)$ is said to be continuous at a if

- $\lim_{x \to a} f(x)$ exists
- $f(a)$ is defined
- $\lim_{x \to a} f(x) = f(a)$.

Review on Calculus

Definition (Differentiability)

$f(x)$ is differentiable at a if

$$\lim_{x \to a} \frac{f(x) - f(a)}{x - a} = f'(a)$$

exists. $f'(a)$ is called the derivative of $f(x)$ at $x = a$.

Theorem

A function $f(x)$ that is differentiable at a is also $f(x)$ continuous at a.

Theorem (Rolle Theorem)

Suppose $f \in C[a,b]$, differentiable on (a,b), and if $f(a)=f(b)$, then there exists a number $c \in (a,b)$ such that $f'(c)=0$.

Theorem (Mean Value Theorem)

If $f(x) \in C[a,b]$ and differentiable on (a,b), then there exists a number $c \in (a,b)$ such that

$$f'(c) = \frac{f(b)-f(a)}{b-a}.$$

Definition (Riemann Integral)

The Reimann integral of a function $f(x)$ over an interval $[a, b]$ is defined as

$$\int_a^b f(x)\,dx = \lim_{\max\{\Delta x_i\}\to 0} \sum_{i=1}^{n} f(z_i)\Delta x_i$$

where $a = x_0 < x_1 < x_2 < \cdots < x_n = b$ and $\Delta x_i = x_i - x_{i-1}$, for each $i = 1, 2, \ldots, n$, and z_i is arbitrary chosen from $[x_{i-1}, x_i]$.

Outline

Decimal, binary and hexadecimal

Decimal (Base 10):

- $4629 = 4 \times 10^3 + 6 \times 10^2 + 2 \times 10^1 + 9 \times 10^0$
- $31.72 = 3 \times 10^1 + 1 \times 10^0 + 7 \times 10^{-1} + 2 \times 10^{-2}$
- In general, $(a_n \dots a_0.b_1 \dots)_{10} = \sum_{k=0}^{n} a_n 10^k + \sum_{k=1}^{\infty} b_n 10^{-k}$

Binary (Base 2):

- $10011_2 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 1 \times 2^1 + 1 \times 2^0 = 19_{10}$
- $11.001_2 = 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-3} = 3.125_{10}$.

Hexadecimal (Base 16):

- $2F91_{16} = 2 \times 16^3 + 15 \times 16^2 + 9 \times 16^1 + 1 \times 16^0 = 12177_{10}$

Decimal, binary and hexadecimal

In Matlab, the conversion on binary or hexa to decimal can be done by

Matlab

```
1 >> bin2dec('10011')
2 >> hex2dec('2F91')
3 >> bin2dec('11.001')
4 >> dec2bin(49)
5 >> dec2hex(49)
```

The line no. 3 will returns NaN. What does it means?

Conversion: Base-10 → Base-2

Example

$3781_{10} = 1110\ 1100\ 0101_2$

$$\begin{array}{r|l}
2 & \underline{3781} \\
2 & \underline{1890}\ 1 \\
2 & \underline{\ 945}\ 0 \\
\vdots &
\end{array}$$

Example

$3781.372_{10} = \dots$

The previous examples tell us that "simple" number in Base-10 is not necessary "simple" in Base-2.

THE END