Chapter 1: Numbers and Expressions

Topics:

- numbers like 2, 3.75, π and $\sqrt{19}$
- evaluation of expressions like

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

roots of
$ax^2 + bx + c = 0$

- representation and approximation of numbers and expressions
- computational errors, can they be avoided or at least controlled?

1.1 Numbers

Numbers for Computations

- integers, rational, real and complex numbers

$-, +, 0$ whole numbers $\leftarrow$ $\mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$

- evaluate expressions
- for solution of equations
  - linear systems of equations $\mathbb{Q}, \mathbb{R}, \mathbb{C}$
  - polynomial equations $\mathbb{C}$
- for continuous functions: $\mathbb{R}$ and $\mathbb{C}$

$2x = 3$

$x^2 = 4/5$

$x^2 + 5 = 0$

Definition: Ring

- A ring R is a set with two binary operations $+$ and $*$ with the following properties
- $(R, +)$ is an abelian group which satisfies, for all $a, b, c \in R$:
  - $a + (b + c) = (a + b) + c$, *associative law*
  - $a + b = b + a$, *commutative law*
  - there exists $0 \in R$ such that $a + 0 = a$
  - there exists $-a$ such that $a + (-a) = 0$
- $(R, *)$ is a monoid where for all $a, b, c \in R$:
  - $a * (b * c) = (a * b) * c$
  - there exists $1 \in \mathbb{R}$ such that $1 * a = a$
- distributive law

$$(a + b) * c = a * c + b * c, \quad a * (b + c) = a * b + a * c$$

Rings in Computation

- all the number sets considered are rings including $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{C}$
- the sets of functions considered are rings including
  - continuous functions
  - polynomials

non-commutative multiplication

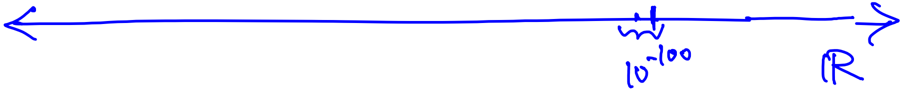
- set of square n by n matrices with elements from a ring is a ring
- the arithmetic laws lead to efficient expression evaluation
- the distributive law is the basis for fast algorithms like fast matrix multiplication, the FFT but also for machine learning and dynamic programming
  - note that the number of multiplications is 2 in $a * b + a * c$ but 1 in $a * (b + c)$

$\mathbb{Q}$ and $\mathbb{R}$

- $\mathbb{Q}$ is a countable subset of $\mathbb{R}$
- $\mathbb{R}$ is used for theory but (subsets of) $\mathbb{Q}$ used for actual computations
- $\mathbb{Q}$ is dense in $\mathbb{R}$, i.e. $\forall x \in \mathbb{R}, \epsilon > 0, \exists u \in \mathbb{Q}$:

$$|x - u| \leq \epsilon$$

$\pi = 3.1415\ldots\ldots$



Decimal and binary fractions

$\{ \frac{p}{q} \mid p \,\&\, q$ are int. $q \neq 0 \}$

$||$

- for computations we consider subsets of $\mathbb{Q}$ for a fixed positive $B \in \mathbb{Z}$

$$\mathbb{Q}_B = \{p\,/B^k \mid p, k \in \mathbb{Z}, k \geq 0\}$$

- here we only consider $B = 10$ or $B = 2$:
  - decimal fractions $\mathbb{Q}_{10}$ used for manual computations
  - binary fractions $\underline{\underline{\mathbb{Q}_2}}$ implemented in computer hardware
- we use the decimal point, e.g.,

$\frac{7}{50} = \frac{14}{100} \in \mathbb{Q}_{10}$

$$44.78 = \frac{4478}{10^2} \in \mathbb{Q}_{10}$$

$2/3 \notin \mathbb{Q}_{10}$

Fractions

$$\mathbb{Z} = \mathbb{Q}_1 \subset \mathbb{Q}_2 \subset \mathbb{Q}_{10} \subset \mathbb{Q}$$

- $\mathbb{Q}_2$ (and $\mathbb{Q}_{10}$) is a dense subset of $\mathbb{Q}$ and thus of $\mathbb{R}$. In particular, each real number can be written as an infinite decimal or binary fraction.
- The sets of fractions $\mathbb{Q}_B$ are all rings and contain the integers

Proposition $\mathbb{Q}_2 \subset \mathbb{Q}_{10}$

Proof.

- $x \in \mathbb{Q}_2$, thus
$$x = \frac{p}{2^k}$$
for some integers p and $k \geq 0$.
- consequently
$$x = \frac{5^k p}{10^k} \in \mathbb{Q}_{10}$$

$$3.1579385791 = \frac{31579385791}{10^{10}}$$

So far: $\mathbb{Q}$ & $\mathbb{R}$ are too hard for computers.

Better: $\mathbb{Q}_2$ or $\mathbb{Q}_{10}$ ↪ .... decimal digits.

↓

Finitely many binary digits

1.2 Representation of Numbers

Simple representation of Integers

- small integers
  - counts: $|, ||, |||, ||||, \ldots$, each number is the cardinality of a set
  - digits: $0, 1, 2, \ldots, 9$, each number has a symbol
  - roman numbers: $I, II, III, IV, \ldots, XXXII, \ldots$
- numbers need to be represented in order to do arithmetics
- all computers (including us) are finite

```
# implementing your own numbers in Python using strings --

x = '||||'
y = '||'

z = x + y   # concatenation is addition

print("OUTPUT:")

print('sum x + y = {}'.format(z))

n = len(z)  # conversion to ordinary integers
print('sum in decimal number system x+y = {}'.format(n))

u = 13*'|'  # conversion to our system
print('conversion of 13:  {}'.format(u))
```

format specification.

```
OUTPUT:
sum x + y = ||||||
sum in decimal number system x+y = 6
```

Representation of Integers

INT.

DECIMAL $213 = 3\times10^{0} + 1\times10^{1} + 2\times10^{2}$

BINARY $101 = 1\times2^{0} + 0\times2^{1} + 1\times2^{2}$

- computer and human number representations are similar, and of the form

$$n = \pm \sum_{k=0}^{t} n_k\, B^k \;=\; \pm\, n_0 n_1 \ldots n_k$$

B: *basis* (humans: $B = 10$, computers: $B = 2$)

NOT INT.

$16.243 = 1\times10^{1} + 6\times10^{0} + 2\times10^{-1} + 4\times10^{-2} + 3\times10^{-3}$

$10.11 = 1\times2^{1} + 0\times2^{0} + 1\times2^{-1} + 1\times2^{-2} = \frac{11}{4}$

The Polynomials which represent integers

- polynomial of degree n

$$p(x) = c_0 + c_1 x + \cdots + c_n x^n$$

- set P of all polynomials is an infinite dimensional linear (vector) space with basis $1, x, x^2, \ldots$
- is also a ring with multiplication defined by

$$p * q\,(x) = p(x)q(x)$$

  - a ring which is also a vector space is called an *algebra*
- for representing the positive integer $n = n_0 + n_1 B + \cdots n_t B^t$ choose the polynomial

$$p(x) = n_0 + n_1 x + \cdots + n_t x^t$$

  and one has

$$n = p(B)$$

- example $n = 739$ and $B = 10$:

$$p(x) = 9 + 3x + 7x^2, \quad \text{thus} \quad p(10) = 739$$

Representation of Rationals

- rationals $\overset{z}{\cancel{x}} \in \mathbb{Q}$ are representated as pairs of integers (p, q) and written as

$$z \quad \cancel{x} = \frac{p}{q} \qquad\qquad \frac{13}{24}$$

- uniqueness is achieved by choosing the $\gcd(p, q) = 1$ (use Euclid's algorithm)
- as rational numbers are ratios of integers, and each integer is represented by a polynomial and a basis B, each rational number $x \in \mathbb{Q}$ is represented by a a rational function

$$r(x) = \frac{p(x)}{q(x)} = \frac{3 \cdot x^0 + 1 \cdot x^1}{4 \cdot x^0 + 2 \cdot x^1}$$

and

$$z \quad \cancel{x} = r(B) = \frac{3+x}{4+2x} \qquad x=10$$

```python
# Rational numbers in Python

from fractions import Fraction

x = Fraction(16, -10)

print("OUTPUT:")

print(x)

y = Fraction('3/7')
print(y)

z = Fraction('1.2341')
print(z)
```

```
OUTPUT:
-8/5
3/7
12341/10000
```

Representation of decimal and binary (and other) fractions

standard integer format

- any $q \in Q_B$ is of the form

$$q = \frac{n}{B^k}$$

for some integers n and k

- example – approximation of $1/3$:

$$\frac{1}{3} = 0.333333\ldots$$

$$\frac{333}{1000}$$

- uses the rational function

$$r(x) = \frac{n_0 + n_1 x + \cdots n_t x^t}{x^k}$$

$$q = \frac{\text{numerator}}{B^k} = \frac{n_0 B^0 + \dots + n_t B^t}{B^k}$$

$$= n_0 B^{-k} + n_1 B^{1-k} + \dots + n_t B^{t-k}$$

set $e = t-k$

scientific format

- any $q \in Q_B$ is of the form

$$q = (n_t + n_{t-1}B^{-1} + \cdots + n_0 B^{-t})B^e$$

$$q = B^e (n_t + n_{t-1}B^{-1} + \dots + n_0 B^{-t})$$

non zero

- example – approximation of $1/3$:

$$0.333$$

- uses the rational function

$$f(x0) = (n_t + n_{-1}x^{-1} + \cdots + n_0 x^{-t})\, x^e$$

where $e = t - k$

$e = -3$

Ex. $0.00125 = 10^{-3}(\underline{\underline{1.25}})$

Representation of Real Numbers

- real numbers $x \in \mathbb{R}$ are represented as (potentially infinite) power series in the basis

  $$x = \pm \sum_{j=-\infty}^{t} n_j B^j$$

  again, the basis $B = 10$ is used in human computation and $B = 2$ is used by computers
- the digits $n_j \in \{0, \ldots, B-1\}$
- real numbers need to be approximated

Representation of Complex Numbers

- complex numbers $z \in \mathbb{C}$ are represented as pairs of reals

$$z = x + iy$$

- addition like vectors
- complex multiplication
- conjugate complex
- imaginary unit i

```python
## Complex numbers in Python

z = 4.0 + 5.0j

print("OUTPUT:")

print("Re(z) = {zr:g}, Im(z) = {zi:g}"
      .format(zr=z.real,zi=z.imag))

I = 1j     # sqrt(-1)

print("I**2 = {}".format(I**2))
```

I^2

```
OUTPUT:
Re(z) = 4, Im(z) = 5
I**2 = (-1+0j)
```

Floating Point Numbers – approximations of real numbers

The set of floating point numbers with t digits to base B is

$$\mathbb{F}_B(t) = \{x = \pm \sum_{j=1}^{t} c_j B^{-j+e} \mid e \in \mathbb{Z}, c_j \in \mathbb{Z}_B, c_1 \neq 0\} \cup \{0\}$$

Ex. $B = 10$ $t = 3$

where $\mathbb{Z}_B = \{0, \ldots, B-1\}$.

$3.41 \in F_{10}(3)$

$\parallel$

$1 \times 10^{-2} + 4 \times 10^{-1} + 3 \cdot 10^{0}$

$123 + 0.345$

$= 123.345 \notin \mathbb{F}_{10}(3)$

$$\mathbb{F}_B(t) \subset \mathbb{Q}_B \subset \mathbb{Q} \subset \mathbb{R}$$

$3.041 \notin F_{10}(3)$

$\in F_{10}(4)$

- $\mathbb{F}_B(t)$ is *not*
  - a ring
  - dense in $\mathbb{Q}$
- motivation: subset of $\mathbb{F}_B(t)$ with $e = e_{\min}, \ldots, e_{\max}$ is computationally feasible
- challenge: floating point arithmetic has to be (re-)defined

Need to store: e, $c_1, c_2, \ldots, c_t$; $\pm$

64 bit number $\to$ 53 digits (binary)

$\mathbb{F}_2(53) \subset \mathbb{Q}_2$

```
## Floating point numbers in Python

x = 0.7
print("OUTPUT:")
print("computer approximation \nx= {0:2.53g}".format(x))

OUTPUT:
computer approximation
x= 0.6999999999999999555910790149937383830547332763671875
```

The IEEE standard 754 floating point numbers

- $B = 2$ and $t = 53$, each number is stored in 64 bit
- special numbers are: 0, $\pm\infty$, some non-normalised numbers, NaNs
- the exponents $e = -1022, \ldots, 1023$
- the standard also specifies details about the arithmetic

Representation of floating point numbers

- representation of any floating point number

$$x = \pm 0.d_1 d_2 \dots d_t \ \cdot B^e$$

  B: *base*, $d_j \in \{0, \dots, B-1\}$: *digits*, e: *exponent*, *number of digits* t

- written as a sum

$$x = sB^e \sum_{j=1}^{t} d_j B^{-j}$$

  $s = +1, -1$ sign

- normalised $d_1 \neq 0$
- $\mathbb{F}\lessdot(B, t)$ denotes floating point numbers with t digits in base B (here we allow any $e \in \mathbb{Z}$, in practice it is a finite range, see notes)

$\mathbb{F}_B(t) =$

IEEE 754 standard on representation

- most commonly used system today: IEEE double precision

number system	base B	number of digits t	exponent range
IEEE double precision	2	53	[-1022, 1023]
IEEE single precision	2	24	[-126, 127]

- Note: there are many other formats, both decimal and binary

Effect of finite exponent

There is a smallest (srictly positive) floating point number

- normalised: $x_{\text{min}} = B^{e_{\text{min}}-1}$
- denormalised: $B^{e_{\text{min}}-t}$

There is a largest (positive) floating point number *
$x_{\text{max}} = B^{e_{\text{max}}} - B^{e_{\text{max}}-t}$
Errors occur when computations exceed these limits

- *underflow* occurs for $0 < x < x_{\text{min}}/2$
- *overflow* occurs when numbers exceed x_{max} We ignore these types of errors in the remainder by allowing $e \in \mathbb{Z}$

Questions

Why do we need numbers? Where do they occur in your experience? Which numbers are most important? What is a number???