Operations, representations

Lecture 3

Recall that we can compute with any finitely representable objects, at least in principle

- Objects from any algebraic system, and even some ad-hoc mixtures
- With any algorithms
  - All steps specified precisely
  - Terminating
- Some processes are not-necessarily terminating. E.g. L'hôpital's rule … use termination heuristics: time/space limits, losing some credibility for results ☹

The Usual Operations

- Integer and Rational:
  - Ring and Field operations +- * exact quotient, remainder
- GCD, factoring of integers
- Approximation via root-finding
- Polynomial operations
  - Ring, Field, GCD, factor
  - Truncated power series
  - Solution of polynomial systems
- Matrix operations (add determinant, resultant, eigenvalues, etc.)

More Operations

- Sorting (e.g. of monomials)
- Union (collections of objects)
- Tests for zero
- Extraction of parts (polynomial degree, constant coefficient, leading coefficient)
- Conversion to different forms ("expand", express algebraic function in a minimal extension, "simplify")

Yet More Operations

Differentiate

Integrate

Limit

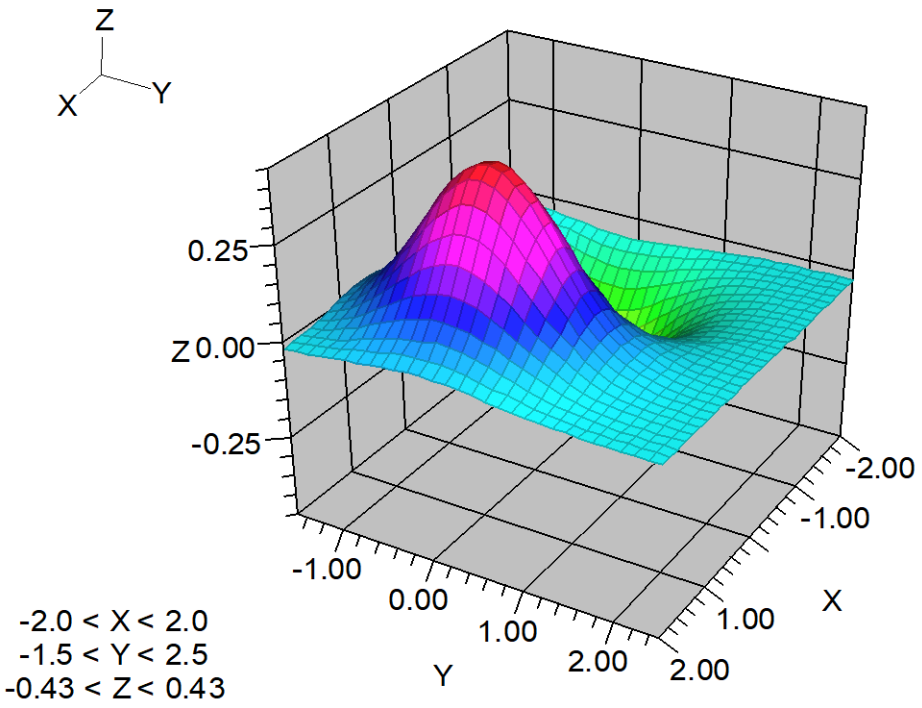
Prove

Find region in which (in)equalities hold

Confirm (as: steps in a proof)

Yet More Operations

Plot



plot3d(exp(-x^2-y^2)*x,x,-2,2,y,-1.5,2.5)

Yet More Operations

Typesetting

$$\text{factor}\left(\text{determinant}\left(\begin{bmatrix} 1 & x & x^2 \\ 1 & y & y^2 \\ 1 & z & z^2 \end{bmatrix}\right)\right) = (y - x)\,(z - x)\,(z - y)$$

Integer representations, operations

- The ring operations +*-
- Euclidean Domain: quotient & remainder, GCD
- UFD : factorization

Unfortunately computers don't do these operations directly

Addition modulo $2^{31}-1$ is rarely what we need.

How do we do arbitrary precision integer arithmetic? (If we could do this, we could build the rationals, and via intervals or some other construction, we could make reals)

Is it hard to do arbitrary integer (bignum) arithmetic?

- In spite of your belief that you are familiar with this subject, there are subtleties. Famous examples: factorization; even in long division!
- You must choose fast algorithms (moderate size) or asymptotically optimal algorithms (large size): what's your target?
- You need fast arithmetic to compute billions of digits of π e.g. GMP or DH Bailey's home page
- Arguably, there are sensitive geometric predicates that require very high precision.

Who cares about integer arithmetic?

- You need to be able to compute with all lengths of numbers to build a computer algebra system: without it your system lies. [SMP used floats; e.g. represented 1/3 by 0.3333333..4.]
- Every Common Lisp has bignum arithmetic built in, some import GMP.

Some ideas just for representing integers

Integers are sequences of characters, 0..9.

Integers are sequences of words modulo 10^9 which is the largest power of 10 less than 2^{31}. [Maple!]

Integers are sequences of hexadecimal digits.

Integers are sequences of 32-bit words storing 16 bits.

Integers are sequences of 32-bit words.

Integers are sequences of 64-bit double-floats (with 8 bits wasted).

Sequences are linked lists

Sequences are vectors

Sequences are stored in sequential disk locations

Sign-magnitude representations possible too.

Yet more ideas

Integers are sequences of 64-bit double-floats (with 8 bits used to position the bits)

e.g. $2^{-300}+2^{300}$ takes 2 words

Integers are stored in redundant form a+b+…

Integers are stored in p-adic form as a sequence of x mod p, x mod p^2, …

Addition in each of these representations

The fastest is the p-adic one, since all the arithmetic can be done without carry, in parallel.

Not usually used because

(a) You can't tell for sure if a number is +, -

(b) Parallelism is almost always irrelevant ☹

(c) If you must see the answer converted to decimal, the conversion is $O(n^2)$

(d) Conversion to decimal may be very common if your application is a bignum calculator.

Multiplication

Extremely well-studied.

The usual method takes $O(n^2)$,

Karatsuba style $O(n^{1.585})$

or FFT style $O(n \log n)$.

These will be studied in the context of multiplying polynomials.

Note that 345 can be mapped to $p(x)=3x^2+4x+5$ where $p(10)$ is 345.

Except for the "carry", the operation is the same.

Integer Division

- This is too tedious to present in a lecture.
- Techniques for guessing the next big digit (bigit) of a quotient within +1 are available, Knuth's *Art of Computer Programming* vol 2 has details.
- For exact division (not div+remainder) consider Newton iteration as an alternative
- FFT / fast multiplication helps

GCD

- Euclid's algorithm for integers is $O(n^2 \log n)$ but is hard to beat in practice, though see analysis of HGCD (Yap) for an $O(n \log^2 n)$ algorithm..
- HGCD is portrayed as a winner for polynomials, but only by complexity analysts who (I suspect, in this case) assume that certain costs are constant when they grow exponentially, and/or subproblems, even small ones, can be done in $O(n \log n)$ time when n^2 is "faster"

Reminder… A Ring R is Euclidean

If there is a function ψ

$$\varphi : R \to \{-\infty\} \cup \mathbb{R}$$

such that

i) $b \neq 0$ and $a|b$ implies $\varphi(a) \leq \varphi(b)$;

ii) for all $r \in \mathbb{R}$, the set $\{\varphi(a) : a \in R, \varphi(a) < r\}$ is finite;

iii) for all $a, b \in R$ $(b \neq 0)$, there exists $q, r \in R$ such that

$$a = bq + r \quad \text{and} \quad \varphi(r) < \varphi(b).$$

Shows the tendency to obfuscate…

What Rings do we use, and what is ψ?
For integers, absolute value ψ
For polynomials in x, degree in x ψ

Where next?

- We could spend a semester on integer arithmetic, but this does not accomplish any higher goals of CAS
- We proceed to polynomials, typically with integer coefficients or finite field coeffs.