Advanced Computing: Numerics and Algebra

Stefan Weinzierl

Institut für Physik, Universität Mainz

I. Introduction

II. Data Structures

III. Efficiency

IV. Classical Algorithms

Literature

Books:

- D. Knuth, “The Art of Computer Programming”, Addison-Wesley, third edition, 1997
- K. Geddes, S. Czapor and G. Labahn, “Algorithms for Computer Algebra”, Kluwer, 1992
- J. von zur Gathen and J. Gerhard, “Modern Computer Algebra”, Cambridge University Press, 1999

Lecture notes:

- S.W., “Computer Algebra in Particle Physics”, hep-ph/0209234.

The need for precision

Hunting for the Higgs and other yet-to-be-discovered particles requires accurate and precise predictions from theory.

Theoretical predictions are calculated as a power expansion in the coupling. Higher precision is reached by including the next higher term in the perturbative expansion.

State of the art:

- Third or fourth order calculations for a few selected quantities (R-ratio, QCD β-function, anomalous magnetic moment of the muon).
- Fully differential NNLO calculations for a few selected $2 \to 2$ and $2 \to 3$ processes.
- Automated NLO calculations for $2 \to n$ ($n = 4..6, 7$) processes.

Computer algebra programs are a standard tool !

History

The early days, mainly LISP based systems:

1965	MATHLAB	1958	FORTRAN
1967	SCHOONSHIP	1960	LISP
1968	REDUCE		
1970	SCRATCHPAD, evolved into AXIOM		
1971	MACSYMA		
1979	muMATH, evolved into DERIVE		

Commercialization and migration to C:

1981	SMP, with successor MATHEMATICA	1972	C
1988	MAPLE		
1992	MuPAD		

Specialized systems:

1975	CAYLEY (group theory), with successor MAGMA
1985	PARI (number theory calculations)
1989	FORM (particle physics)
1992	MACAULAY (algebraic geometry)

A move to object-oriented design and open-source:		1984	C++
1999	GiNaC	1995	Java

The naive wish-list related to computer algebra systems

- Completeness: Covers every branch of modern mathematics
- Intelligence: Finds a solution we never dreamt of
- Efficiency in performance: Compilation
- Short development cycles: Interactive use and high-quality output
- Support for modern program paradigmas: Object-oriented
- Standardized programming language: Development tools exist
- Source code freely available: Portability, bug hunting

Unfortunately, these wishes contradict each other.

What computer algebra system to choose ?

The choice depends on the specific needs:

- Local problems:
  Problem expands into a sum of different terms and each term can be solved independently of the others.
  Complications: The number of terms can become quite large.
  Requirements: Bookkeeping, ability to handle large amounts of data.

- non-local problems:
  Standarized non-local operations (e.g. factorization).
  Requirements: An implementation of these algorithms.

- non-standard problems:
  Dedicated algorithms, developed by the user to solve a specific problem.
  Requirements: Ability to model abstract mathematical concepts in the programming language of the computer algebra system

Requirements on a computer algebra system

Computer-intensive symbolic calculations in particle physics can be characterized by:

- Need for basic operations like addition, multiplication, sorting ...
- Specialized code usually written by the user
- No need for a system which knows “more” than the user!

CAS on the market:

- Commercial: Mathematica, Maple, Reduce, ...
- Non-commercial: FORM, GiNaC, ...

Data structures

1. Lists
2. Containers
3. Object-oriented design: A little bit C++
4. A very simple computer algebra system

Symbolic differentiation

Symbolic differentiation can be specified by a few rules:

$$
\begin{aligned}
\frac{d}{dx}c &= 0, \\
\frac{d}{dx}x &= 1, \\
\frac{d}{dx}(f(x)+g(x)) &= \frac{d}{dx}f(x)+\frac{d}{dx}g(x), \\
\frac{d}{dx}(f(x)g(x)) &= \left(\frac{d}{dx}f(x)\right)g(x)+f(x)\left(\frac{d}{dx}g(x)\right).
\end{aligned}
$$

These rules are sufficient to differentiate polynomials in x.

LISP

LISP is based on lists and uses the prefix notation:

$$a+b+c \quad \widehat{=} \quad (+\ a\ b\ c)$$

LISP takes the first entry in a list as the name of an operation and applies this operation to the remaining elements.

A single quote ’ in front of a list prohibits evaluation.

A backquote ‘ acts like a single quote, except that any commas that appear within the scope of the backquote have the effect of unquoting the following expression.

A free LISP interpreter is available from http://www.gnu.org/software/gcl/gcl.html.

Symbolic differentiation in LISP

```
(DEFUN OPERATOR (LIST) (CAR LIST))

(DEFUN ARG1 (LIST) (CADR LIST))

(DEFUN ARG2 (LIST) (CADDR LIST))

(DEFUN DIFF (E X)
  (COND ((ATOM E) (COND ((EQUAL E X) 1)
                        (T 0)))
        ((EQUAL (OPERATOR E) '+)
         `(+ ,(DIFF (ARG1 E) X) ,(DIFF (ARG2 E) X)))
        ((EQUAL (OPERATOR E) '*)
         `(+ (* ,(DIFF (ARG1 E) X) ,(ARG2 E))
             (* ,(ARG1 E) ,(DIFF (ARG2 E) X))))))
```

Example

LISP is an interactive language and entering

```
(DIFF '(* A X) 'X)
```

at the prompt for

$$\frac{d}{dx}(ax)$$

yields

```
(+ (* 0 X) (* A 1))
```

which stands for

$$(0 \cdot x) + (a \cdot 1).$$

Simplifications like $0 \cdot x = 0$, $a \cdot 1 = a$ or $0 + a = a$ are out of the scope of this simple example.

Remarks

- Distinction between atoms and containers.
- The use of recursive techniques.
- Lists are used to represent data structures.
- Lists can be nested.
- The output is not necessarily in the most compact form.

Containers

Container: an object that holds other objects (lists, arrays, ...).

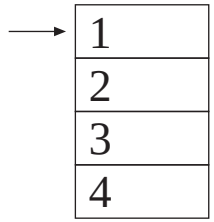
Various possibilities how the data can be stored in physical memory.
The time needed to access one specific element will depend on the lay-out of the data in the memory.

$O(1)$	cheap
$O(\log(n))$	fairly cheap
$O(n)$	expensive
$O(n\log(n))$	expensive
$O(n^2)$	very expensive
$O(e^n)$	unaffordable
$O(n!)$	unaffordable

Operations of order $O(1)$ or $O(\log(n))$ are considered “cheap” operations.
Considerable speep-up, if an operation which naively takes $O(n^2)$ time, can be improved to $O(n\log(n))$.
In generally one tries to avoid operations, which take $O(n^2)$ time.

Arrays

Arrays (also called “vectors” in C++):



Data stored in consecutive slots in memory

Given the address of the first element and the size of a single entry, the address of the j’th element is

$$\text{addr}_j \quad = \quad \text{addr}_1 + (j-1)\,\text{size}$$

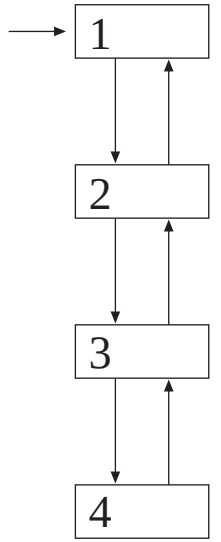
Access: $O(1)$

Insertion: $O(n)$

Lists

Often implemented as double-linked list.

Each node contains: an entry
pointer to the previous node
pointer to the next node



Access: $O(n)$

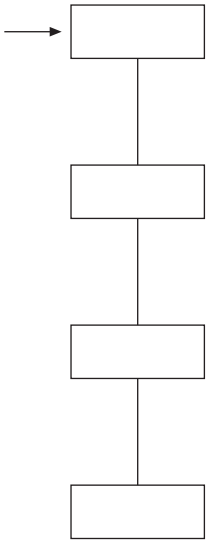
Insertion: $O(1)$

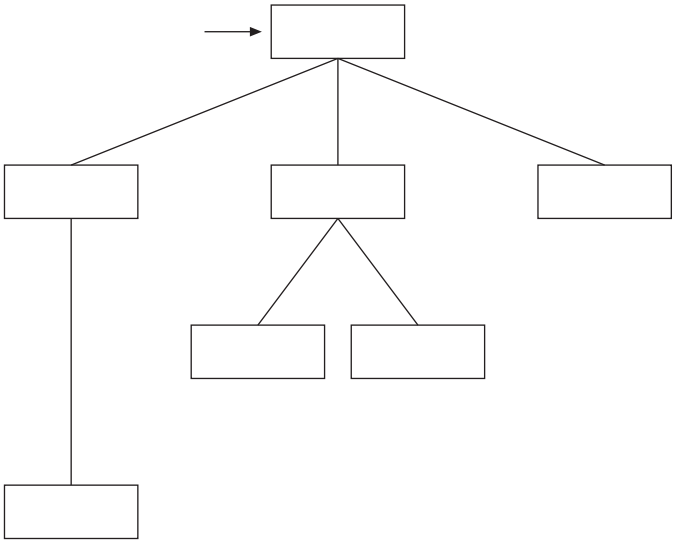
Rooted trees

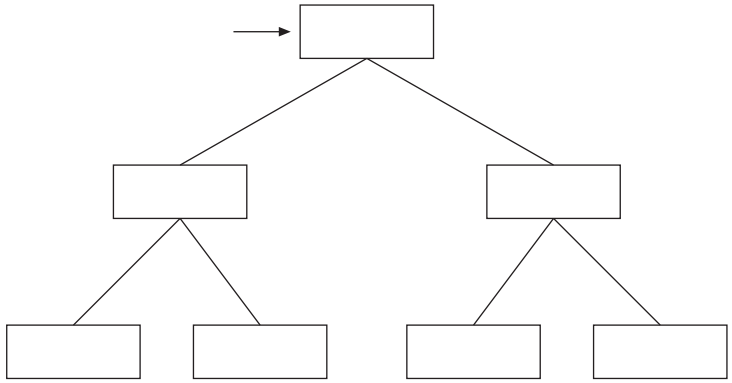
Linear structure:
Lists

Generalization:
Rooted trees

Specialization:
Binary trees





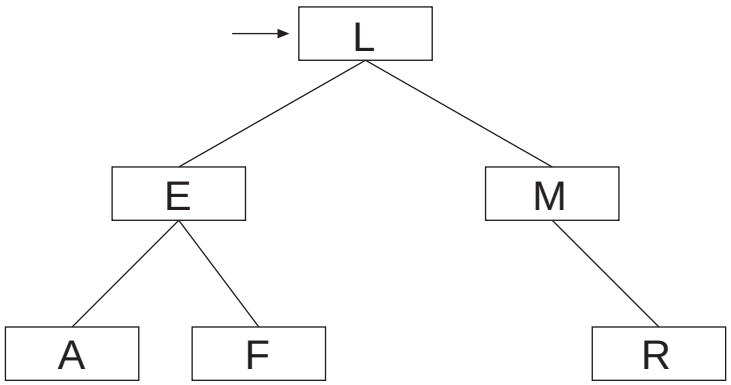


Associative arrays

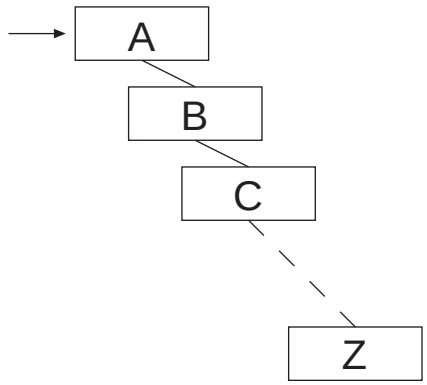
Store pairs (key,value), for the keys: “less-than” operation

Example: Name / Telephone number

Luca
Matteo
Roberto
Enrico
Francesco
Alberto



Anna
Beate
Cecilia
Dora
...
Zoe



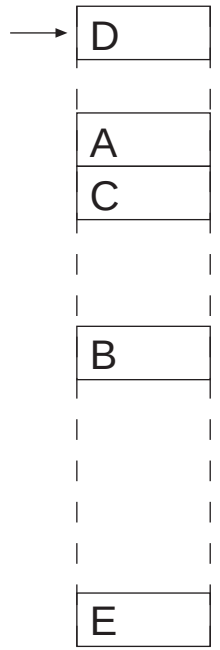
Access: $O(\log(n))$
Insertion: $O(\log(n))$

Hash maps

Hash maps store again pairs (key,value).

For the keys: - a hash function
- “is-equal” operation

The hash function is used to compute from the key the address.
The “is-equal” operation is used to check for collisions.
The efficiency depends on the quality of the hash function.



Access: $O(1)$

Insertion: $O(1)$

Summary on containers

	implementation	subscription	list operations	back operations
vector	one-dimensional array	$O(1)$	$O(n)$	$O(1)$
list	double-linked list	$O(n)$	$O(1)$	$O(1)$
map	binary tree	$O(\log(n))$	$O(\log(n))$	—
hash map	hash map	$O(1)$	$O(1)$	—

Object-oriented programming

Object-oriented techniques invented for development and maintenance of large software projects.

Languages: C++, Java;

C++ allows operator overloading:

```
fourvector v,w;
v + w;
```

Java doesn't:

```
fourvector v,w;
v.add_vec(w);
```

Operator overloading makes programs more readable, specially in science.

C++ in 3 seconds

C++ supports object-oriented programming:

- Classes consists of data members and member functions operating on the data.
- Separation of the implementation from the interface by private and public members.
- Modelling of similar concepts through derived classes and inheritance.
- Polymorphic behaviour through virtual functions
- Run-time type information
- Backwards compatible with C.

Modular approach

Split the program into independent entities.

In C++: classes

Distinguish between information other modules have to know about the class (“header files”) and implementations of methods, whose details can be hidden.

Example: complex numbers

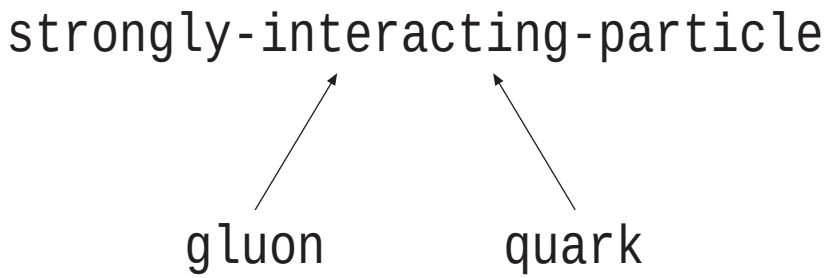
```
class complex {
 public:
  double norm();

 private:
  double x,y;
};
```

Data abstraction

Can define new data types: complex numbers, Feynman diagrams, ...

Inheritance: Model similar concepts through derived classes in a class hierarchy:



Virtual functions

Re-use and extension of a given program:

traditional: new code calls old code

object-oriented: old code calls new code

```
class employee
{
 virtual void transfer_salary();
}

class professor : public employee
{
 void transfer_salary();
};

class secretary : public employee
{
 void transfer_salary();
};

void pay_salary(employee * name)
 {
  name->transfer_salary();
 }
```

```
class junior_professor : public professor
{
 void transfer_salary();
};
```

Generic algorithms: Templates

Example: Sorting a list of integers (`int`) or real numbers (`double`). The algorithm is the same and does only require a “less-than” operation.

Templates allow to code this algorithm, where the data type (`int`, `double`, ...) is variable.

The C++ Standard Template Library (STL) offers a wide range of data types and algorithms (`vector`, `list`, `map`, ...).

Data storage

- static: Data is stored at a fixed address during the complete execution of the program.

- automatic on the stack: For temporary variables in a subroutine, storage space is created on the stack when entering the subroutine and deleted automatically when leaving the subroutine.

- dynamically on the heap: Memory can also be allocated dynamically on the heap from everywhere inside the program. If it is not freed when it is not needed any more, it blocks the memory until the program ends.

A simple computer algebra system

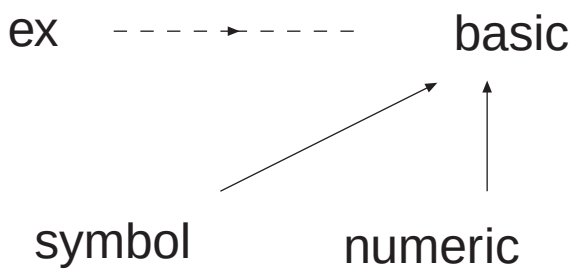
Should know about: symbols $a, b, c, \ldots$
integers $1, 2, 3, \ldots$
addition, multiplication

and simple evaluation rules like

$$5a + 3b + 2a \quad \longrightarrow \quad 7a + 3b$$

Objects: Expressions
atomic: symbols, numbers
containers: add, mul

Class structure




```
class symbol: public basic
{
 protected:
  std::string name;
};

class numeric : public basic
{
 protected:
  int value;
};
```

One long expression can be pointed to by more than one pointer.

```
class ex
{
 public:
  basic * bp;
};
```

The base class “basic”

Objects of type “basic” or derived types can potentially contain large amounts of data.

Allocate the memory dynamically on the heap.

Use reference counting:

```
class ex
{
 public:
  basic * bp;
};

class basic
{
 protected:
  unsigned refcount;
};
```

Copy-on-write semantics

- Since an instance of the class “ex” consists basically only of a pointer, it is extremely light-weight.

- The object pointed to is reference counted. No copying takes place in the following lines of code:

```
ex e1 = 3*x + 42;   // refcount is 1
ex e2 = e1;         // refcount is 2
```

- Copying is necesarry when one expression is changed:

```
e2 = e2 + 1;        // refcount of 3*x + 42 is 1
                    // refcount of 3*x + 43 is 1
```

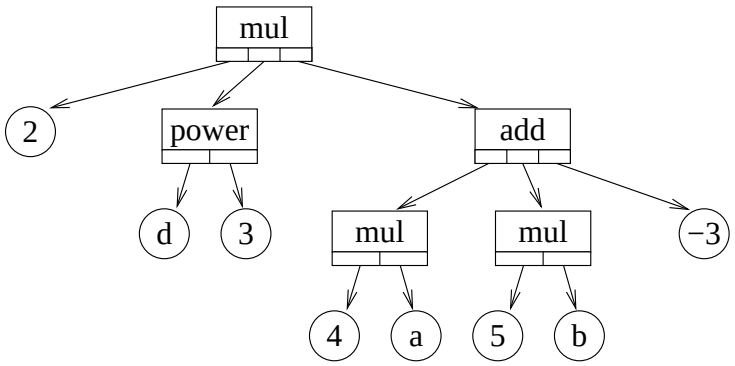
- Simple garbage collection: If an object is no longer referenced, it is deleted and the memory is freed.

Container classes

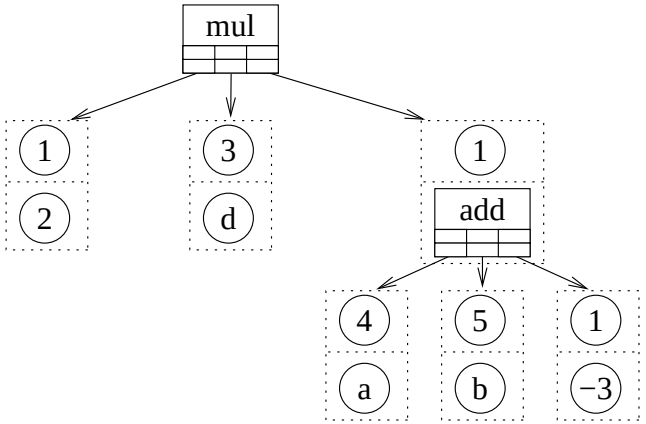
Containers can be nested, but deep trees are inefficient.

$$2d^3\,(4a+5b-3)$$

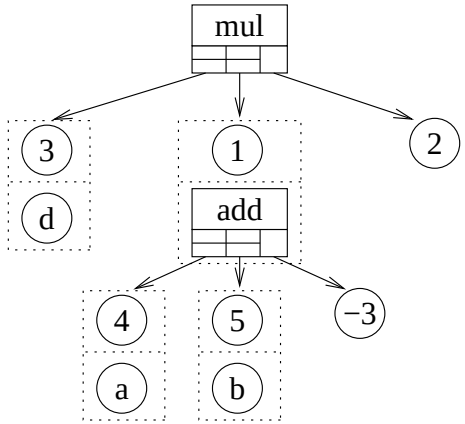
naive:



pairs (number, rest):



in reality:



Container classes

Data representation of

$$3a+2b+5c \quad \text{and} \quad a^3b^2c^5$$

is identical: $(3,a)\ (2,b)\ (5,c)$

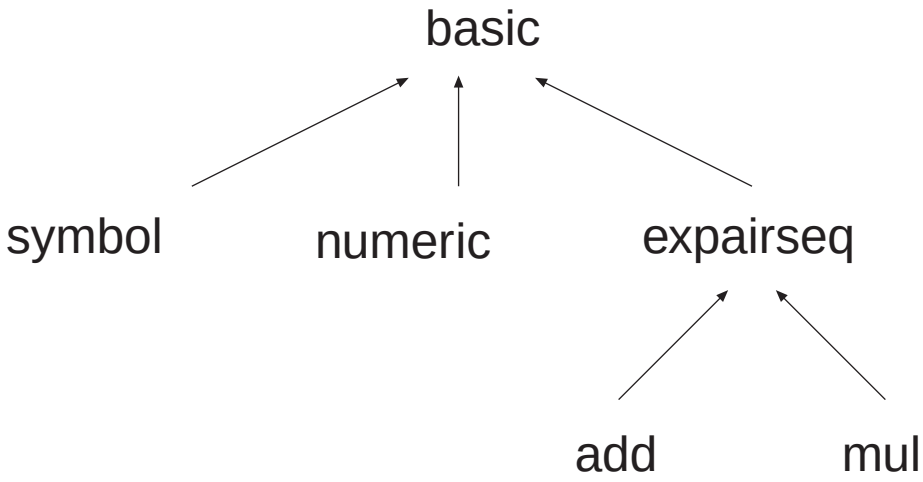
Use common base class:

```
class expairseq : public basic
{
 protected:
  std::vector<expair> seq;
}
```

The containers `add` and `mul` are then derived from `expairseq`.

Class structure

All classes are derived from an abstract base class “basic”:



In addition there is a “smart pointer” to these classes:

The user deals only with the class “ex”.

Virtual functions make sure that the appropriate method is called.

Automatic evaluation

Implementation of some simple evaluations like:

$$5a + 3b + 2a \quad \longrightarrow \quad 7a + 3b$$

Evaluation happens at the first assignment of an object:

- The class `basic` has a status flag `evaluated`.
- Every time an object is assigned, this flag is checked.
- If the flag is not set, the appropriate method `eval()` is called.

  In the example above, `eval()` does first a sorting: $(2, a)$, $(5, a)$, $(3, b)$.

  Second step: items with identical second entry are combined: $(2, a)$, $(5, a) \longrightarrow (7, a)$.

GiNaC

The GiNaC library (GiNaC is Not a CAS) allows symbolic calculations in C++.

The Standard Template Library for C++ offers already good support for the manipulation of lists and vectors.

GiNaC does not try to provide extensive algebraic capabilities and a simple programming language
but instead accepts a given language (C++) and extends it by a set of algebraic capabilities.

available at: `http://www.ginac.de`

Summary

- CPU time depends on the representation of the data in the memory.
- Object-oriented techniques for large projects.
- Structure of a simple computer algebra system.

Efficiency

1. Recursion
2. Multiplication of large numbers
3. The greatest common divisor

Recursion

The Fibonacci numbers:

$$f(1) = 1, \quad f(2) = 1,$$
$$f(n) = f(n-1) + f(n-2).$$

As a program:

```
ex fibonacci(int n)
{
 if ( (n == 1) || (n == 2) ) return 1;

 return fibonacci(n-1) + fibonacci(n-2);
}
```

Suppose we want to know $f(1000)$. How often does this program then call $f(10)$?

Unrolling the recursion

```
ex fibonacci(ex n)
{
 if ( n==1 ) return 1;

 std::vector<ex> f(n);

 f[0] = 1; // the first entry is f[0]
 f[1] = 1;
 for (int j=2; j<n; j++)
  {
   f[j] = f[j-1] + f[j-2];
  }
 return f(n-1);
}
```

Now every value is calculated exactly once.

The Fibonacci numbers

Formula from Binet:

$$f(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Generating function:

$$\frac{1}{1-z-z^2} = \sum_{n=0}^{\infty} f(n)\, z^n,$$

$$f(n) = \sum_{k=0}^{n} \binom{n-k}{k}.$$

Multiplication of large numbers

Let $(a_0, a_1, \ldots, a_{n-1})$ represent a number with n digits in base B:

$$a = a_{n-1}B^{n-1} + \ldots + a_1 B + a_0$$

Can also write

$$a = a_h B^{\frac{n}{2}} + a_l$$

a_h and a_l have $n/2$ digits.

Multiplication (as in primary school):

$$ab = a_h b_h B^n + (a_h b_l + a_l b_h) B^{n/2} + a_l b_l$$

Requires n^2 one-digit multiplications: $O(n^2)$

Karatsuba multiplication

Addition costs less than multiplication.

Rewrite:

$$ab = a_h b_h B^n + [a_h b_h + a_l b_l - (a_h - a_l)(b_h - b_l)] B^{n/2} + a_l b_l$$

Requires only 3 multiplications of integers with $n/2$ digits.

Grows like

$$O(n^{\log_2 3}) \approx O(n^{1.58})$$

with the number of digits n.

Schönhage-Strassen multiplication

Can consider the sequence $(a_0, a_1, \ldots, a_n)$ as defining a polynomial:

$$a(x) \quad = \quad a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n.$$

Polynomial of degree n is uniquely defined by the values at $n+1$ distinct points:

$$(a(x_0), a(x_1), \ldots, a(x_n)) \quad \text{(modular representation)}$$

Polynomial $a(x)$ of degree n, $b(x)$ of degree m. Then the product is of degree $n+m$.

$$a(x) = (a(x_0), \ldots, a(x_{n+m})), \qquad b(x) = (b(x_0), \ldots, b(x_{n+m})).$$

Then

$$a(x)b(x) \quad = \quad (\, a(x_0) \cdot b(x_0),\ \ldots,\ a(x_{n+m}) \cdot b(x_{n+m})\,), \quad \text{only } O(n+m) \text{ operations!}$$

In the modular representation, multiplication of two polynomials of degree n is an $O(n)$ operation.

Fast Fourier transform

Need to convert between the standard representation and the modular one.

Standard $\longrightarrow$ modular: Freedom of choice for the points $x_0, x_1, \ldots$. Choose

$$\left\{1, \omega, \omega^2, \ldots, \omega^{n-1}\right\},$$

where ω is a primitive n-th root of unity, e.g. $\omega^n = 1$, but $\omega^k \neq 1$ for $0 < k < n$.
Nice feature:

$$\omega^{i+\frac{n}{2}} = -\omega^i, \qquad \left(\omega^{i+n/2}\right)^2 = \left(\omega^i\right)^2$$

Write the polynomial $a(x)$ in the form

$$a(x) \quad = \quad \underbrace{b(x^2)}_{\text{even}} + \underbrace{x \cdot c(x^2)}_{\text{odd}},$$

Need to evaluate $b(x^2)$ and $c(x^2)$ only at $n/2$ distinct points instead of n.
If $n = 2^m$: $O(n \log n)$.

CLN, NTL and GMP

CLN, NTL and GMP are libraries to handle large numbers. GiNaC is built on CLN:

- Memory effiency: Small integers immediate, garbage collection
- Speed effiency: Assembler language kernel, Karatsuba and Schönhage-Strassen multiplication

available at: `http://www.ginac.de/CLN`

NTL is a powerful library for the factorization of univariate polynomials.

available at: `http://www.shoup.net/ntl`

GMP is written in C.
available at: `http://gmplib.org`

The greatest common divisor

Consider

$$\frac{(x+y)^2(x-y)^3}{(x+y)(x^2-y^2)} \quad = \quad (x-y)^2.$$

For humans: One factor of $(x+y)$ is trivially removed.
For remaining factors note that $(x^2-y^2) = (x+y)(x-y)$.

For a computer algebra system:

- Factorization into irreducible polynomials is very expensive and actually not required.

- To cancel the common factors it is sufficient to calculate the greatest common divisor (gcd) of the two expressions.

- The efficient implementation of an algorithm for the calculation of the gcd is essential for many other algorithms.

Polynomial algebra

Most gcd calculations are done in polynomial rings.

A ring $(R, +, \cdot)$ is a set R with two operations $+$ and $\cdot$, such that:

- $(R, +)$ is an abelian group
- multiplication is associative and distributive

We also assume that

- R is commutative
- has a unit element for the multiplication

Rings

Commutative ring: example $\mathbb{Z}_8$, in this ring $2 \cdot 4 = 0$

↑

Integral domain: example $\{a + bi\sqrt{5} | a, b \in \mathbb{Z}\}$, here $21 = 3 \cdot 7 = (1 - 2i\sqrt{5})(1 + 2i\sqrt{5})$

↑

Unique factorization domain: example $\mathbb{Z}[x]$

↑

Euclidean domain: division with remainder $a = bq + r$, example $\mathbb{Z}$

↑

Field: every non-zero element has an inverse: $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}_p$

Polynomial rings

Structure of polynomial rings in one variable and several variables depending on the underlying coefficient ring R:

R	$R[x]$	$R[x_1, x_2, \ldots, x_n]$
commutative ring	commutative ring	commutative ring
integral domain	integral domain	integral domain
unique factorization domain	unique factorization domain	unique factorization domain
euclidean domain	unique factorization domain	unique factorization domain
field	euclidean domain	unique factorization domain

The algorithm of Euclid

In an Euclidean domain: $a = bq + r$ with $r < b$. It follows:

$$\gcd(a,b) \quad = \quad \gcd(b,r).$$

Proof: Let $c = \gcd(a,b)$ and $d = \gcd(b,r)$.

c divides r, since $r = a - bq$; therefore c divides d.

d divides a, since $a = bq + r$; therefore d also divides c.

Algorithm: Set

$$r_0 = a, \quad r_1 = b \quad \text{and} \quad r_i = \text{rem}(r_{i-2}, r_{i-1}) \quad \text{until} \quad r_{k+1} = 0.$$

Then

$$\gcd(a,b) \quad = \quad \gcd(r_0,r_1) = \gcd(r_1,r_2) = \ldots = \gcd(r_{k-1},r_k) = r_k.$$

Example

Let us compute $\gcd(21,6)$:

$$\begin{array}{llll} r_0 = 21, & r_1 = 6, & r_0 = 3r_1 + 3, & r_2 = 3, \\ r_1 = 6 & r_2 = 3, & r_1 = 2r_1 + 0, & r_3 = 0. \end{array}$$

Therefore

$$\gcd(21,6) \quad = \quad 3.$$

gcd in polynomial rings

Polynomial rings usually only unique factorization domains, but not Euclidean domains:

$$a(x) = x^2 + 2x + 3 \in \mathbb{Z}[x], \qquad b(x) = 5x + 7 \in \mathbb{Z}[x].$$

Division with remainder not possible:

$$a(x) \quad \neq \quad b(x)q(x) + r(x), \quad \text{with} \quad q(x), r(x) \in \mathbb{Z}[x].$$

But in $\mathbb{Q}[x]$:

$$a(x) \quad = \quad \left(\frac{1}{5}x + \frac{3}{25}\right) b(x) + \frac{54}{25}$$

Obstruction arrises from the leading coefficent of $b(x)$.

Extension of the Euclidean algorithm

Introduce a pseudo-division with remainder. Let

$$\begin{aligned} a(x) &= a_n x^n + \ldots + a_0, \\ b(x) &= b_m x^m + \ldots + b_0 \end{aligned}$$

with $n \geq m$ and $b(x) \neq 0$. There exists $q(x), r(x)$ such that

$$b_m^{n-m+1} a(x) = b(x)q(x) + r(x)$$

with $\deg(r(x)) < \deg(b(x))$.

This pseudo-divison property is sufficient to extend the Euclidean algorithm to polynomial rings over unique factorization domains.

Drawback

Intermediate expressions can become quite long:

$$\begin{aligned} a(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, \\ b(x) &= 3x^6 + 5x^4 - 4x^2 - 9x + 21, \end{aligned}$$

Calculate the pseudo-remainder sequence:

$$\begin{aligned} r_2(x) &= -15x^4 + 3x^2 - 9, \\ r_3(x) &= 15795x^2 + 30375x - 59535, \\ r_4(x) &= 1254542875143750x - 1654608338437500, \\ r_5(x) &= 1259333879550074310093114199218750 0. \end{aligned}$$

This implies that $a(x)$ and $b(x)$ are relatively prime, but the numbers which occur in the calculation are large.

Subresultant polynomial remainder sequence

Can avoid large numbers, if each polynomial is split into a content part and a primitive part:

$$r_3 \quad = \quad 15795x^2 + 30375x - 59535 = 1215\left(13x^2 + 25x + 49\right)$$

Find balance between: - keep expressions small
- avoid extra gcd calculation in the coefficient domain

Subresultant polynomial remainder sequence:

$$c_i^{\delta_i+1} r_{i-1}(x) \quad = \quad q_i(x) r_i(x) + d_i r_{i+1}(x)$$

c_i leading coefficient of $r_i(x)$.

Heuristic gcd algorithm

Example:

$$a(x) = 6x^4 + 21x^3 + 35x^2 + 27x + 7, \qquad b(x) = 12x^4 - 3x^3 - 17x^2 - 45x + 21.$$

Evaluate at $\xi = 100$: $a(100) = 621352707$ and $b(100) = 1196825521$.
The gcd of these two numbers is

$$c \quad = \quad \gcd(621352707, 1196825521) = 30607.$$

Write 30607 in ξ-adic representation:

$$30607 \quad = \quad 3 \cdot 100^2 + 6 \cdot 100 + 7.$$

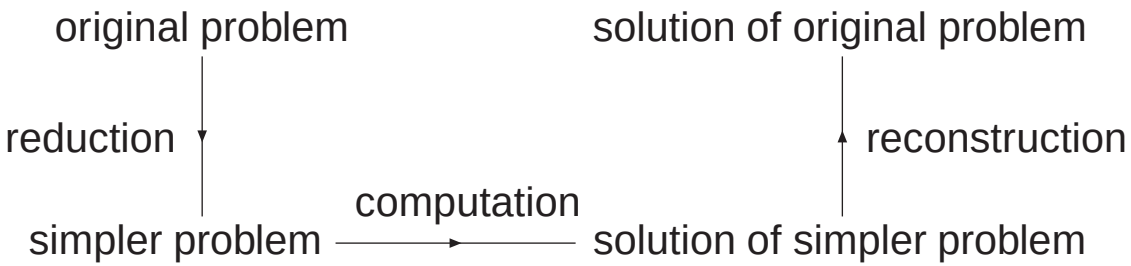
Candidate for gcd is $g(x) = 3x^2 + 6x + 7$.

Theorem: If ξ is chosen large enough and $g(x)$ divides $a(x)$ and $b(x)$, then $g(x) = \gcd(a(x), b(x))$.

Summary on efficieny

Divide and conquer: - associative arrays with binary trees
- Karatsuba multiplication
- Fast Fourier transform

Solve simpler problem first:



Classical algorithms

1. Factorization
2. Symbolic integration
3. Gröbner bases

Factorization

Factorization of $u(x) \in \mathbb{Z}[x]$ due to Berlekamp:

a) Factor out the gcd of the coefficients and perform a square-free decomposition.

b) Factor the polynomial in the ring $\mathbb{Z}_p$, where p is a prime number. If p were sufficiently large, the factorization over $\mathbb{Z}$ could be read off from the factorization over $\mathbb{Z}_p$.

c) Lifting: Construct a factorization over $\mathbb{Z}_{p^r}$ from a factorization over $\mathbb{Z}_p$.

State-of-the-art:
- Cantor-Zassenhaus
- Kaltofen-Shoup
- NTL-library (Shoup)

Square-free decomposition

Suppose a polynomial $u(x)$ contains a factor $v(x)$ to some power:

$$u(x) = [v(x)]^m r(x).$$

Take the derivative

$$u'(x) = m[v(x)]^{m-1} v'(x) r(x) + [v(x)]^m r'(x)$$

and calculate the gcd

$$g(x) = \gcd\left(u(x), u'(x)\right).$$

$[v(x)]^{m-1}$ is a factor of the gcd. Therefore

$$u(x) = \left(\frac{u(x)}{g(x)}\right) g(x).$$

Computational cost: one (or several) gcd calculation(s).

Berlekamp's algorithm

After square-free decomposition:

$$u(x) \quad = \quad p_1(x)p_2(x)\ldots p_k(x), \quad \deg u(x) = n.$$

Factorization in $\mathbb{Z}_p$: Define the entries $q_{k,j}$ of a $n \times n$ matrix Q by

$$x^{kp} \quad = \quad \left(q_{k,n-1}x^{n-1} + \ldots + q_{k,1}x + q_{k,0}\right) \bmod u(x).$$

Solve

$$(v_0, v_1, \ldots, v_{n-1})\, Q \quad = \quad (v_0, v_1, \ldots, v_{n-1})$$

This defines $v(x) = v_{n-1}x^{n-1} + \ldots + v_1 x + v_0$.
Calculate

$$\gcd\left(u(x), v(x) - s\right), \quad 0 \leq s < p.$$

This will detect the non-trivial factors of $u(x)$ in $\mathbb{Z}_p$.

Hensel lifting

Have: $u(x) = v_1(x)w_1(x) \bmod p$, want: $u(x) = v_r(x)w_r(x) \bmod p^r$.

$u(x)$ square-free $\Rightarrow$ $\gcd(v_1, w_1) = 1 \bmod p$ $\Rightarrow$ can find $a(x)$, $b(x)$ such that

$$a(x)v_1(x) + b(x)w_1(x) = 1 \bmod p, \quad \deg a < \deg w_1, \quad \deg b < \deg v_1.$$

Step from (v_r, w_r) to (v_{r+1}, w_{r+1}):

compute c_r :	$p^r c_r(x)$	$= v_r(x)w_r(x) - u(x)$	mod p^{r+1}
compute q_r, a_r :	$a(x)c_r(x)$	$= q_r(x)w_1(x) + a_r(x)$	mod p
compute b_r :	$b_r(x)$	$= b(x)c_r(x) + q_r(x)v_1(x)$	mod p
compute v_{r+1}, w_{r+1} :	$v_{r+1}(x)$	$= v_r(x) - p^r b_r(x)$	mod p^{r+1},
	$w_{r+1}(x)$	$= w_r(x) - p^r a_r(x)$	mod p^{r+1}.

Example

Factorization of $u(x) = x^2 + 27x + 176 \in \mathbb{Z}[x]$.

Step 1: $u(x)$ is already square-free.

Step 2: Factorization in $\mathbb{Z}_3$:

$$\begin{aligned} u(x) &= x^2 + 2 \bmod 3 \\ &= (x+1)(x+2) \bmod 3. \end{aligned}$$

Step 3: Hensel lifting

$$\begin{aligned} u(x) &= (x+7)(x+2) \bmod 9 \\ &= (x+16)(x+11) \bmod 27. \end{aligned}$$

The factorization in $\mathbb{Z}_{27}$ agrees already with the factorization in $\mathbb{Z}$.

Symbolic integration

Example: Integration of rational functions

$$f(x) = \frac{a(x)}{b(x)} = \frac{a_m x^m + \ldots + a_1 x + a_0}{b_n x^n + \ldots + b_1 x + b_0}.$$

Suppose we know the factorization of $b(x)$ over $\mathbb{C}$:

$$b_(x) = b_n (x - c_1)^{m_1} \ldots (x - c_r)^{m_r},$$

After polynomial division and partial fraction decomposition:

$$f(x) = p(x) + \sum_{i=1}^{r} \sum_{j=1}^{m_i} \frac{d_{ij}}{(x - c_i)^j},$$

where $p(x)$ is a polynomial in x and the d_{ij} are complex numbers.
The integration of $p(x)$ is trivial.

Symbolic integration

For the pole terms we have

$$\int \frac{dx}{(x-c_i)^j} = \begin{cases} \ln(x-c_i), & j=1, \\ \frac{1}{(1-j)}\frac{1}{(x-c_i)^{j-1}}, & j>1. \end{cases}$$

Major inconvenience: Need to factor the denominator completely and thereby introduce algebraic extensions (like square roots or complex numbers), which drop out in the final result:

$$\frac{1-x^2}{(1+x^2)^2} = -\frac{1}{2}\frac{1}{(x+i)^2} - \frac{1}{2}\frac{1}{(x-i)^2},$$

but

$$\int dx \frac{1-x^2}{(1+x^2)^2} = \frac{x}{1+x^2}.$$

Symbolic integration

Better: - compute as much as possible in the domain of the integrand
- find the minimal algebraic extension necessary to express the integral

Step 1: Hermite's reduction method

Step 2: Rothstein-Trager algorithm

Hermite's reduction method (analog to square-free decomposition):

$$f(x) \quad = \quad p(x) + \frac{a(x)}{b(x)}, \quad \text{with} \quad \deg a < \deg b \quad \text{and} \quad \gcd(a,b) = 1.$$

Let $b(x) = u(x)\,[v(x)]^m$. Compute polynomials r and s (Euclid's algorithm) such that

$$r(x)u(x)v'(x) + s(x)v(x) \quad = \quad \frac{1}{1-m}a(x).$$

Then we obtain for the integral

$$\int dx \frac{a(x)}{u(x)\,[v(x)]^m} \quad = \quad \frac{r(x)}{[v(x)]^{m-1}} + \int dx \frac{(1-m)s(x) - u(x)r'(x)}{u(x)\,[v(x)]^{m-1}}.$$

The algorithm by Rothstein and Trager

We are left with an integral of the form

$$\int dx \frac{a(x)}{b(x)}$$

with deg $a <$ deg b and b is square-free.

The result will be

$$\int dx \frac{a(x)}{b(x)} = \sum_{i=1}^{n} r_i \ln(x - c_i)$$

where the c_i's are the zeros of b and the r_i's are the residues at the c_i's.

An efficient algorithm to determine the c_i and r_i was invented by Rothstein and Trager and later improved by Trager, Lazard and Rioboo.

The Risch integration algorithm

Integration of rational functions $\Rightarrow$ generalization to elementary functions.

Elementary functions: rational functions
logarithms
exponentials
algebraic functions (square roots, ...)

Risch algorithm: Given elementary function f

- decides whether $\int dx\ f(x)$ can be expressed as elementary function,
- if so, constructive method.

If this is not the case, we know at least that the integral cannot be expressed in terms of elementary functions.

Gröbner bases

Motivation: Simplification with respect to side relations

$$s_j(x_1,\ldots,x_k) = 0, \quad j = 1,\ldots,r.$$

Want to write

$$f \quad = \quad a_1 s_1 + \ldots + a_r s_r + g,$$

where g is "simpler" than f.

The precise meaning of "simpler" requires the introduction of an order relation.

Example: Lexicographic ordering, e.g. x is "more complicated" as y, and x^2 is "more complicated" than x.

Gröbner bases

Example: consider the expressions

$$f_1 = x + 2y^3, \qquad f_2 = x^2,$$

which we would like to simplify with respect to the siderelations

$$s_1 = x^2 + 2xy^2, \qquad s_2 = xy + 2y^3 - 1.$$

Naive approach: - take each siderelation,
- determine its “most complicated” element,
- replace each occurence in f by the simpler terms of the siderelation.

Example: Simplification of f_2 with respect to s_1 and s_2:

$$f_2 = x^2 = s_1 - 2xy^2 = s_1 - 2ys_2 + 4y^4 - 2y,$$

and f_2 would simplify to $4y^4 - 2y$.

Gröbner bases

Expressions:

$$f_1 = x + 2y^3, \qquad f_2 = x^2.$$

Siderelations:

$$s_1 = x^2 + 2xy^2, \qquad s_2 = xy + 2y^3 - 1.$$

But: If s_1 and s_2 are siderelations, any linear combination is again a siderelation:

$$s_3 \quad = \quad ys_1 - xs_2 = x$$

is a siderelation which can be deduced from s_1 and s_2.

Therefore f_2 simplifies to 0.

Gröbner bases

Consider multivariate polynomials in the ring $R[x_1, \ldots, x_k]$.
Each element can be written as a sum of monomials of the form $cx_1^{m_1} \ldots x_k^{m_k}$.

Define a lexicographic order of these terms by

$$cx_1^{m_1} \ldots x_k^{m_k} > c' x_1^{m'_1} \ldots x_k^{m'_k},$$

if the leftmost nonzero entry in $(m_1 - m'_1, \ldots, m_k - m'_k)$ is positive.

Can write any element $f \in R[x_1, \ldots, x_k]$ as

$$f = \sum_{i=0}^{n} h_i \quad \text{with} \quad h_{i+1} > h_i.$$

Leading term:

$$\mathrm{lt}(f) = h_n$$

Ideals

Let $B = \{b_1, \ldots, b_r\}$ be a finite set of polynomials. The set

$$\langle B \rangle = \langle b_1, \ldots, b_r \rangle \quad = \quad \left\{ \sum_{i=1}^{r} a_i b_i \,\middle|\, a_i \in R[x_1, \ldots, x_k] \right\}$$

is called the ideal generated by the set B.

The set B is also called a basis for this ideal.

Denote by $\mathrm{lt}(B)$ the set of leading terms of B.

Gröbner bases

Consider an ideal I generated by the finite set G:

$$I \quad = \quad \langle G \rangle .$$

G is a basis for I. G is called a Gröbner basis, if in addition

$$\langle \mathrm{lt}(G) \rangle \quad = \quad \langle \mathrm{lt}(I) \rangle$$

Algorithm to compute a Gröbner basis by Buchberger.

Example

$f_1 = x + 2y^3, \quad f_2 = x^2, \quad s_1 = x^2 + 2xy^2, \quad s_2 = xy + 2y^3 - 1.$

$\{s_1, s_2\}$ is not a Gröbner basis, since $\mathrm{lt}(s_1) = x^2$ and $\mathrm{lt}(s_2) = xy$ and

$$\mathrm{lt}(ys_1 - xs_2) = x \notin \langle \mathrm{lt}(s_1), \mathrm{lt}(s_2) \rangle.$$

A Gröbner basis is given by $\{b_1, b_2\}$, where

$$b_1 = x, \qquad b_2 = 2y^3 - 1.$$

f_1 and f_2 can be written as:

$$\begin{aligned} f_1 &= b_1 + b_2 + 1, \\ f_2 &= xb_1 + 0, \end{aligned}$$

e.g. f_1 simplifies to 1 and f_2 simplifies to 0.

Summary

- Structures
  - Containers: Arrays, lists, associative arrays, hash maps, ...
  - Design for a large programming project: modular
  - Structure of a simple computer algebra system
- Efficiency
  - CPU time depends on the representation of the data in the memory
  - Efficiency of algorithms: Divide and conquer
    Solve simpler problem first
- Algorithms
  - Factorization
  - Symbolic integration
  - Gröbner bases