

GiNaC

Symbolic computation with C++

Jens Vollinga

Institut für Physik
Universität Mainz



C++ library
Developed since 1999

- CAS in high energy physics
- Fundamentals of GiNaC
- Features of GiNaC
- Applications

Perturbation theory

- Feynman rules
- Feynman diagrams
- Dirac algebra and traces
- Loop integration
- Summation of diagrams
- Phase space integration
- Checks
- ...

Perturbation theory

- Feynman rules
- Feynman diagrams
- Dirac algebra and traces
- Loop integration
- Summation of diagrams
- Phase space integration
- Checks
- ...

Combinatorics

Algebra

⇒ Data management

Numerics

I/O

Perturbation theory

- Feynman rules
- Feynman diagrams
- Dirac algebra and traces
- Loop integration
- Summation of diagrams
- Phase space integration
- Checks
- ...

Combinatorics

(Maple, C, Perl)

Algebra

(Maple, FORM)

⇒ Data management

(C, BASH, Perl)

Numerics

(FORTRAN, C)

I/O

(C, Tcl/Tk)

Taming COMPLEXITY → Software Engineering
(coding, social, technology, . . .)

Taming COMPLEXITY → Software Engineering
(coding, social, technology, . . .)

What if you are a C++ programmer?

- Combinatorics
- Algebra
- Data management
- Numerics
- I/O

Taming COMPLEXITY → Software Engineering
(coding, social, technology, . . .)

What if you are a C++ programmer?

- ~~Combinatorics~~
- Algebra
- ~~Data management~~
- Numerics
- I/O

Taming COMPLEXITY → Software Engineering
(coding, social, technology, . . .)

What if you are a C++ programmer?

- ~~Combinatorics~~
- Algebra
- ~~Data management~~
- ~~Numerics~~
- I/O

Taming COMPLEXITY → Software Engineering
(coding, social, technology, . . .)

What if you are a C++ programmer?

- ~~Combinatorics~~
- Algebra
- ~~Data management~~
- ~~Numerics~~
- ~~I/O~~

Taming COMPLEXITY → Software Engineering
(coding, social, technology, . . .)

What if you are a C++ programmer?

- ~~Combinatorics~~
- Algebra → GiNaC
- ~~Data management~~
- ~~Numerics~~
- ~~I/O~~

- Write a C++ program:

```
#include <iostream>
using namespace std;
#include <ginac/ginac.h>
using namespace GiNaC;

int main()
{
    symbol x("x");
    ex result = Li(2,x).diff(x);
    cout << result << endl;
    return 0;
}
```

Fundamentals of GiNaC

■ Write a C++ program:

```
#include <iostream>
using namespace std;
#include <ginac/ginac.h>
using namespace GiNaC;

int main()
{
    symbol x("x");
    ex result = Li(2,x).diff(x);
    cout << result << endl;
    return 0;
}
```

Library header

Fundamentals of GiNaC

- Write a C++ program:

```
#include <iostream>
using namespace std;
#include <ginac/ginac.h>
using namespace GiNaC;
```

```
int main()
```

```
{
```

```
    symbol x("x");
```

```
    ex result = Li(2,x).diff(x);
```

```
    cout << result << endl;
```

```
    return 0;
```

```
}
```

New data types from GiNaC

Fundamentals of GiNaC

■ Write a C++ program:

```
#include <iostream>
using namespace std;
#include <ginac/ginac.h>
using namespace GiNaC;
```

```
int main()
```

```
{
```

```
    symbol x("x");
```

```
    ex result = Li(2,x).diff(x);
```

```
    cout << result << endl;
```

```
    return 0;
```

```
}
```

Algebra in C++ !

Fundamentals of GiNaC

- Compile the program:

```
$ g++ -o myprg myprg.cpp -lginaC
```

- Run it:

```
$ ./myprg  
-log(1-x)*x^(-1)
```


Container for arbitrary algebraic expressions

→ *ex*

Container for arbitrary algebraic expressions

→ `ex`

After declaration

```
ex myexpr;
```

or as a function parameter for example

```
ex nloopfct(ex momentum)
```

```
{ ... }
```

Fundamentals of GiNaC

Container for arbitrary algebraic expressions

→ `ex`

After declaration

```
ex myexpr;
```

or as a function parameter for example

```
ex nloopfct(ex momentum)
{ ... }
```

use it like C++ built-in types:

```
momentum = sqrt(p) * pow(mu, 2) - d;
myexpr = sin(x).series(x, 10);
cout << myexpr << endl;
if (myexpr.has(x)) y = myexpr;
```

Fundamentals of GiNaC

Container for arbitrary algebraic expressions

→ `ex`

After declaration

```
ex myexpr;
```

or as a function parameter for example

```
ex nloopfct(ex momentum)
{ ... }
```

use it like C++ built-in types:

```
momentum = sqrt(p) * pow(mu, 2) - d;
myexpr = sin(x).series(x, 10);
cout << myexpr << endl;
if (myexpr.has(x)) y = myexpr;
```

Symbols \rightarrow symbol

Declaration:

```
symbol x("x");  
symbol eps("\epsilon")
```

Symbols \rightarrow `symbol`

Declaration:

```
symbol x("x");  
symbol eps("\epsilon")
```

What else?

- Numbers 1.34, 3/4, 2i, ...
- Mathematical functions
- Matrices
- Algebras

...

- Arbitrary symbolic expressions

Features of GiNaC

- Arbitrary symbolic expressions
- Complex arithmetics with arbitrary precision

Features of GiNaC

- Arbitrary symbolic expressions
- Complex arithmetics with arbitrary precision
- Operations on rational functions,

e.g. $\frac{x^2 - y^2}{(x + y)^2} \rightarrow \frac{x - y}{x + y}$

Features of GiNaC

- Arbitrary symbolic expressions
- Complex arithmetics with arbitrary precision
- Operations on rational functions,
e.g. $\frac{x^2-y^2}{(x+y)^2} \rightarrow \frac{x-y}{x+y}$
- Symbolic derivation and series expansion

Features of GiNaC

- Arbitrary symbolic expressions
- Complex arithmetics with arbitrary precision
- Operations on rational functions,
e.g. $\frac{x^2-y^2}{(x+y)^2} \rightarrow \frac{x-y}{x+y}$
- Symbolic derivation and series expansion
- Matrices, vectors, linear equation systems

Features of GiNaC

- Arbitrary symbolic expressions
- Complex arithmetics with arbitrary precision
- Operations on rational functions,
e.g. $\frac{x^2-y^2}{(x+y)^2} \rightarrow \frac{x-y}{x+y}$
- Symbolic derivation and series expansion
- Matrices, vectors, linear equation systems
- In-/Output of expressions in various formats
(text, binary, LaTeX, ...)

Features of GiNaC

- Arbitrary symbolic expressions
- Complex arithmetics with arbitrary precision
- Operations on rational functions,
e.g. $\frac{x^2-y^2}{(x+y)^2} \rightarrow \frac{x-y}{x+y}$
- Symbolic derivation and series expansion
- Matrices, vectors, linear equation systems
- In-/Output of expressions in various formats
(text, binary, LaTeX, ...)
- New functions and classes can easily be added

■ Functions

$\Gamma, B, \psi, \binom{n}{k}, \dots$

$\sin, \cos, \tan, \sinh, \dots$

$\sqrt{}, \exp, \log, \zeta, S, H, Li, \dots$

- Functions

$\Gamma, B, \psi, \binom{n}{k}, \dots$

$\sin, \cos, \tan, \sinh, \dots$

$\sqrt{\quad}, \exp, \log, \zeta, S, H, Li, \dots$

- Objects with indices: p^μ, A_{ij}
Special algebras: Clifford, SU(3)

- Functions

$\Gamma, B, \psi, \binom{n}{k}, \dots$

$\sin, \cos, \tan, \sinh, \dots$

$\sqrt{}, \exp, \log, \zeta, S, H, Li, \dots$

- Objects with indices: p^μ, A_{ij}

Special algebras: Clifford, SU(3)

- Automatic code generation (\rightarrow numeric integration)

- Functions

$\Gamma, B, \psi, \binom{n}{k}, \dots$

$\sin, \cos, \tan, \sinh, \dots$

$\sqrt{}, \exp, \log, \zeta, S, H, Li, \dots$

- Objects with indices: p^μ, A_{ij}

Special algebras: Clifford, SU(3)

- Automatic code generation (\rightarrow numeric integration)

- Good documentation, open source (GPL)

\rightarrow www.ginac.de

Applications with GiNaC

- Loop calculations: xloops
- nestedsums
- gTybalt
- feelfem, PURRS, MBDyn
- Publications

On the Invariance of Residues of Feynman Graphs

I. Bierenbaum, R. Kreckel, D. Kreimer, J. Math. Phys. 43 4721-4740 (2002)

The Massless Two-Loop Two-Point Function

I. Bierenbaum, S. Weinzierl, Eur. Phys. J. C32:67-78 (2003)

The Electroweak Standard Model in the Axial Gauge

C. Dams, R. Kleiss, Eur. Phys. J. C34:419-427 (2004)

An Example of Clifford Algebras Calculations with GiNaC

V. Kisil, [arXiv:cs.MS/0410044]

GiNaC – a C++ library for symbolic computation

- Complete computational framework in C++
- Features of GiNaC

GiNaC – a C++ library for symbolic computation

- Complete computational framework in C++
- Features of GiNaC
- Developed since 1999
 - Version 1.0 in 2001, current version 1.3.1
 - Technically matured and well documented
- Active development going on
 - (better factorization, more special functions, improved interface to MC integration)

www.ginac.de