

UNIVERSIDADE FEDERAL DE VIÇOSA  
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS  
DEPARTAMENTO DE INFORMÁTICA

**PROJETO FINAL DE CURSO**

**BIBLIOTECA PARA MATEMÁTICA SIMBÓLICA EM C++**

**Arthur Gonçalves do Carmo**

*Graduando em Ciência da Computação*

Luiz Carlos de Abreu Albuquerque  
(Orientador)

VIÇOSA – MINAS GERAIS

JUNHO – 2018

# **RESUMO**

## **BIBLIOTECA PARA MATEMÁTICA SIMBÓLICA EM C++**

Luiz Carlos de Abreu Albuquerque (Orientador)

Arthur Gonçalves do Carmo (Estudante)

### **RESUMO**

Matemática simbólica é a área da computação que se preocupa com representar e manipular equações e expressões matemáticas de forma simbólica, em oposição aos métodos de manipulação por aproximação numérica. O objetivo do projeto é criar uma biblioteca para C++ contendo classes que representam simbolicamente alguns dos principais objetos matemáticos, como números inteiros e racionais, aritmética modular, polinômios e expressões.

### **PALAVRAS-CHAVE**

matemática simbólica; aritmética de precisão múltipla; expressões matemáticas

### **ÁREA DE CONHECIMENTO**

1.03.02.01- 8 - Matemática Simbólica

### **LINHA DE PESQUISA**

DPI-040 – Algoritmos e Otimização Combinatória

# 1 – Introdução

A matemática simbólica é um campo já muito bem estudado na computação. A dificuldade de se criarem métodos mais eficientes do que os já existentes levam a uma relativa ausência de opções para bibliotecas para aritmética de precisão múltipla e representação simbólica. O mesmo não pode ser dito, entretanto, sobre sistemas de manipulação algébrica simbólica, chamados CAS (*Computer Algebra System*), que são produtos de software completos para o mesmo fim e possuem grande variedade, muitas vezes diferenciando-se um do outro por trabalharem com campos bem distintos de aplicação da matemática.

É fácil notar que a matemática simbólica é mais interessante do ponto de vista teórico do que do prático. Para a maioria das aplicações, as aproximações em ponto flutuante são suficientemente precisas e, quando não são, é mais interessante a aritmética de precisão múltipla do que resultados simbólicos. As aplicações que se beneficiam da matemática simbólica são, em sua maior parte, da área de matemática e de física teórica. Além disso, a matemática simbólica pode ser considerada uma área pertinente aos limites da computação, e também uma forma de analisar a própria abordagem humana em relação à matemática.

## 1.2 – Objetivos

O objetivo geral do trabalho será desenvolver uma biblioteca para matemática simbólica em C++, com classes para representar números inteiros, modulares e racionais, polinômios e expressões matemáticas, derivação e integração simbólicas.

Os objetivos específicos deste trabalho são:

- Estudar técnicas e algoritmos usados em computação simbólica
- Aprimorar e aplicar conhecimentos sobre a linguagem C++
- Utilizar padrões de desenvolvimento para software livre

## 2 – Referencial Teórico

### 2.1 – Números de precisão múltipla

Os computadores modernos representam números inteiros como cadeias de bits, mais comumente 32 ou 64 bits, interpretadas como números inteiros em base 2. Essa abordagem tem a limitação de poder representar, em 64 bits, apenas números entre 0 e  $2^{64}$  (sem usar bit de sinal) ou entre  $-2^{63}$  e  $2^{63}-1$  (com bit de sinal). A ideia da aritmética de precisão múltipla é representar números inteiros cujo tamanho será limitado apenas pela memória disponível no computador [1, 2, 3, 4, 5].

Uma excelente alternativa para trabalhar com números de precisão múltipla é a *GNU Multiple-Precision Library – GMP*, uma biblioteca escrita na linguagem C, altamente otimizada e que trabalha com números inteiros, racionais e ponto flutuante de precisão arbitrária.

É pertinente ao escopo do trabalho a criação de classes para representar números inteiros e racionais de precisão múltipla.

### 2.2 – Expressões algébricas

#### 2.2.1 – Monômios com coeficientes racionais

Monômios são expressões algébricas que consistem apenas da multiplicação entre constantes e variáveis (chamadas literais). Um monômio possui a forma:

$$c * \prod_{i=0}^{\infty} x_i^{k_i}$$

Onde  $c$  é um número racional,  $x_i$  é uma variável única no monômio e  $k_i$  é o expoente ou grau da variável  $x_i$ . O produtório  $\prod_{i=0}^{\infty} x_i^{k_i}$  é chamado a parte literal do monômio.

Um monômio  $M_1$  é semelhante a um monômio  $M_2$  se  $M_1$  e  $M_2$  possuem a mesma parte literal. Nesse caso, a soma entre  $M_1$  e  $M_2$  é também um monômio.

O grau de um monômio é o valor da soma dos expoentes  $k_i$  de sua parte literal.

### 2.2.2 – Polinômios com coeficientes racionais

Polinômios são expressões que consistem em variáveis e coeficientes, envolvendo apenas as operações de adição, subtração, multiplicação e expoentes inteiros não negativos. Um polinômio de grau  $N$  pode ser representado da forma:

$$\sum_{i=0}^N a_i x^i$$

Onde cada coeficiente  $a_i$  é um número racional. É interessante que, para fins de representação, podemos considerar que  $a_i$  possa também ser um polinômio, o que nos leva a uma representação recursiva de polinômios [2, 3] com mais de uma variável:

$$\sum_{i=0}^N \sum_{j=0}^M a_{ij} x^i y^j = \sum_{i=0}^N c_i x^i \quad \text{onde} \quad c_i = \sum_{j=0}^M a_{ij} y^j$$

Outra forma de representar um polinômio é como uma soma de monômios:

$$\sum_{i=0}^{\infty} M_i$$

Nesse caso, o grau do polinômio é grau do monômio  $M_i$  de maior grau. Essa representação já generaliza polinômios com mais de uma variável.

## 2.3 – GNU Build System

O *GNU Build System* é uma convenção para organização, construção (build) e instalação de pacotes de forma padronizada. Idealmente a instalação de um pacote que segue o *GNU Build System* em qualquer ambiente necessita apenas dos comandos: `./configure`, `make`, e `make install`. Usaremos esta convenção para o desenvolvimento do software.

## 3 – Metodologia

### 3.1 – Representação

#### 3.1.1 – Números inteiros

Um número inteiro é representado como uma dupla  $\left( \sum_{i=0}^{N-1} d_i b^i, S \right)$ , onde  $b$  é a base de representação,  $N$  é o número de dígitos do número,  $0 \leq d_i < b$  é o  $(i+1)$ -ésimo dígito menos significativo e  $S$  é sinal do número.

A transcrição dessa representação para uma linguagem de programação consiste em um arranjo onde cada posição armazena um dígito na base escolhida, e um valor booleano para armazenar o sinal. Nesse projeto, os números são representados em *little endian*, isto é, os dígitos mais significativos possuem índices maiores.

A base de representação escolhida para o projeto é a base  $10^9$  porque bases que são potência de 10 facilitam a leitura e escrita dos números em base 10, e alguns dos algoritmos necessitam que o valor do quadrado da base ( $b^2$ ) seja representável por um tipo básico da linguagem. Utilizando um sistema de 64 bits, temos  $(10^9)^2 = 10^{18} < 2^{64} < 10^{20} = (10^{10})^2$ .

Além disso, a classe ainda guarda a quantidade de dígitos (em base  $10^9$ ) do número. O nome da classe será **num\_z**.

#### 3.1.2 – Números racionais

Os números racionais são representados como uma tripla  $(N_1, N_2, S)$ , onde  $N_1$  é um número inteiro não negativo,  $N_2$  é um número inteiro positivo e  $S$  é o sinal do número.

No projeto, a classe dos números racionais utiliza a classe dos números inteiros para representar  $N_1$  e  $N_2$  e um byte para representar  $S$ . Os números são representados em sua forma irredutível. O nome da classe será **num\_q**.

### 3.1.3 – Aritmética modular

Os números usados para a aritmética modular são representados como uma dupla  $(N, B)$ , onde  $N$  é um número inteiro e  $B$  é a base da aritmética. Um número  $x \bmod B$  é o resto da divisão de  $x$  por  $B$ .

No projeto, a classe dos números modulares é uma classe *template* que recebe a base como o *tipo* do *template* e possui um membro que é um número inteiro para armazenar  $N$ . O nome da classe será **num\_zm<B>**.

### 3.1.4 – Tuplas de divisão inteira

O resultado da divisão inteira de um inteiro  $M$  por um inteiro  $N$ , resulta em um quociente  $q$  e um resto  $r$  de forma que  $M = q * N + r$ .

A forma usada para representar a dupla  $(q, r)$ , é por meio de duas estruturas de duplas, chamadas **div\_tuple** (usada para a divisão) e **mod\_tuple** (usada para a operação de resto da divisão).

A estrutura **div\_tuple** armazena o quociente e resto da divisão inteira de  $M$  por  $N$  de forma que o resto da divisão é sempre não negativo, enquanto a estrutura **mod\_tuple** armazena o quociente e resto da divisão inteira de  $M$  por  $N$  de forma que o resto da divisão é zero ou possui o mesmo sinal de  $N$ .

### 3.1.5 – Monômios com coeficientes racionais

Um monômio é representado como uma tripla  $(C, F: K \rightarrow \mathbb{N}, D)$ , onde  $C$  é um número racional,  $F: K \rightarrow \mathbb{N}$ , é a função que associa o valor  $K$  de uma variável ao grau daquela variável no monômio, e  $D$  é o grau do monômio, que é a soma dos graus de todas as variáveis do monômio.

No projeto,  $C$  é um objeto **num\_q**,  $D$  é um objeto **num\_z** e  $F: K \rightarrow \mathbb{N}$  é um container que associa uma **string** que representa uma variável a um objeto **num\_z** que representa o grau daquela variável. O nome da classe será **monomial**.

### 3.1.6 – Polinômios com coeficientes racionais

Um polinômio é representado apenas como um conjunto de monômios, de forma que cada termo do polinômio não é semelhante a nenhum outro termo do mesmo polinômio. O nome da classe será **polynomial**.

## **4 – Produto a ser obtido**

Como resultado final, obteremos uma biblioteca de classes que implementam as operações matemáticas simbólicas para números inteiros, racionais, modulares, polinômios, expressões algébricas, derivação e integração simbólicas.



## 5 – Cronograma

<b>Ações</b>	<b>Ago 2018</b>	<b>Set 2018</b>	<b>Out 2018</b>	<b>Nov 2018</b>	<b>Dez 2018</b>
<b>Especificações de Requisitos</b>	X	X	X		
<b>Implementação de Números</b>	X				
<b>Implementação de Polinômios</b>		X	X		
<b>Implementação de Expressões</b>			X	X	
<b>Implementação de Derivadas</b>				X	
<b>Implementação de Integrais</b>				X	X
<b>Elaboração da Monografia</b>			X	X	X

## Referências

- [1] KNUTH, D. **The Art of Computer Programming: Seminumerical Algorithms**. 2. ed. Reading, Massachusetts: Addison-Wesley, 1998.
- [2] LISKA, R. et al. **Computer Algebra, Algorithms, Systems and Applications**. Disponível em: <<http://www-troja.fjfi.cvut.cz/~liska/ca/>> Acesso em: 17 de setembro de 2018
- [3] COHEN, J. S. **Computer Algebra and Symbolic Computation: Elementary Algorithms**. Natick, Massachusetts: A K Peters, 2002.
- [4] COHEN, J. S. **Computer Algebra and Symbolic Computation: Mathematical Methods**. Natick, Massachusetts: A K Peters, 2003.
- [5] SHI, T. K., STEEB, W. H., HARDY, Y. **SymbolicC++: An Introduction to Computer Algebra Using Object-Oriented Programming**. 2. ed. Londres: Springer-Verlag, 2000.