

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

SUMÁRIO

RESUMO.....	04
1. Introdução.....	05
1.1 Objetivos.....	05
2 Referencial Teórico.....	06
2.1 Números de precisão múltipla.....	06
2.1.1 Números inteiros.....	06
2.1.2 Números racionais.....	06
2.1.3 Números em aritmética modular.....	06
2.2 Expressões algébricas.....	07
2.2.1 Monômios com coeficientes racionais.....	07
2.2.2 Polinômios com coeficientes racionais.....	08
2.2.3 Expressões aritméticas.....	08
3. Metodologia.....	09
3.1 Representação.....	09
3.1.1 Números inteiros.....	09
3.1.2 Números racionais.....	09
3.1.3 Aritmética modular.....	10
3.1.4 Tuplas de divisão inteira.....	10
3.1.5 Monômios com coeficientes racionais.....	10
3.1.6 Polinômios com coeficientes racionais.....	10
3.1.7 Tupla de divisão de polinômios.....	11
3.1.8 Termos básicos.....	11
3.1.9 Funções.....	11
3.1.10 Expressões aritméticas.....	11
3.2 Manipulação.....	12
3.2.1 Métodos comuns.....	12

3.2.2 Métodos especializados.....	12
3.2.2.1 Números inteiros e racionais.....	12
3.2.2.2 Números inteiros.....	12
3.2.2.3 Números racionais.....	12
3.2.2.4 Aritmética modular.....	13
3.2.2.5 Monômios e polinômios.....	13
3.2.2.6 Monômios.....	13
3.2.2.7 Polinômios.....	13
4. Considerações Finais.....	14
5. Referências.....	15
6. APÊNDICE A – Código fonte, instruções de instalação e exemplos de uso da biblioteca.....	16
7. ANEXO A – <i>GNU BUILD SYSTEM</i>	17

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.

1.1 – 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

2.1.1 – Números inteiros

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.

2.1.2 – Números racionais

Os números racionais podem ser descritos como os números x que têm a forma $x = \frac{a}{b}$, com $a, b \in \mathbb{Z}$ e $b \neq 0$.

Uma forma mais interessante quanto à representação dos números racionais, é considerar que $a \in \mathbb{Z}, b \in \mathbb{N} \setminus \{0\}$ e a e b são primos entre si ($\text{mdc}(a, b) = 1$). Um número racional nessa forma está em sua forma irredutível, essa forma é única para cada número racional.

2.1.3 – Números em aritmética modular

Uma aritmética módulo N é uma forma de aritmética para números inteiros na qual os números dão “voltas” ao atingirem o valor N . Um exemplo do dia-a-dia é a forma como as horas são marcadas num relógio de parede que é uma aritmética módulo 12 (considerando que a hora 12 é, na verdade, a hora 0), ou em um relógio em formato de 24 horas, que é uma aritmética módulo 24.

Baseado na descrição acima, dois números são congruentes em uma aritmética modular caso ambos tenham a mesma representação naquela aritmética, por exemplo:

$$19 \bmod 7 \equiv 5 \bmod 7 \quad \text{e} \quad 21 \bmod 3 \equiv 0 \bmod 3$$

De forma geral, um número $x \bmod N$, é o resto da divisão inteira de x por N .

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 [4]:

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

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 [4]:

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

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.

2.2.3 – Expressões aritméticas

Uma expressão pode ser representada por uma gramática livre de contexto (CFG) com as seguintes regras [5]:

expressão \rightarrow termo [+ termo] [- termo]

termo \rightarrow fator [* fator] [/ fator]

fator \rightarrow polinômio | função | (expressão)

Onde polinômio e função são *tokens* que representam as classes de funções e de polinômios do projeto.

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 possui apenas dois membros, uma para representar o número e uma para representar a base modular. O nome da classe será **num_zm**.

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**.

3.1.7 – Tupla de divisão de polinômios

Assim como os números inteiros, a divisão de um polinômio P por um polinômio G gera um quociente q e resto r , de forma que $P = G * q + r$. Para polinômios com coeficientes racionais de mais de uma variável, a equação não necessariamente tem solução única, e o método de divisão utilizado por padrão é o método de divisão por monômios, encontrado em [4].

3.1.8– Termos básicos

Termos básicos de uma expressão podem ser um polinômio ou uma função, dessa forma a classe para termos básicos possui apenas um ponteiro para função e um objeto do tipo polinômio. A classe tem o objetivo de ser invisível para o usuário. O nome da classe será **term**.

3.1.9 – Funções

As funções são tratadas de forma absolutamente simbólica, é representada como uma dupla $(S, Args)$, onde S é uma *string* que representa o símbolo da função, e $Args$ é a lista dos argumentos da função, que são expressões. O nome da classe será **function**.

3.1.10 – Expressões aritméticas

A classe que representa expressões é como um nó de uma árvore binária, uma quintupla da forma $(T^*, LSE^*, RSE^*, +, S)$, onde T^* é um ponteiro para um termo básico, LSE^* é um ponteiro para a expressão que está do lado esquerdo do operador (se houver), RSE^* é um ponteiro para a expressão que está do lado direito do operador (se houver), $+$ é a operação representada na expressão e S é o sinal da expressão. Esta é a principal classe simbólica do projeto, será chamada **Expr**.

3.2 – Manipulação

3.2.1 – Métodos comuns

Todas as classes possuem os operadores aritméticos básicos $(+, -, *, /)$, o método *pow*, para exponenciação por número inteiro e o operador unário $(-)$ para o inverso aditivo.

Todas as classes também possuem os métodos *is_null()* e *is_zero()*, que indicam se a variável tem valor zero.

As classes de números inteiros, racionais, monômios e polinômios possuem métodos para encontrar o maior divisor comum e o mínimo múltiplo comum entre duas instâncias da classe.

Assim como os algoritmos para maior divisor comum e para operações aritméticas $(+, -, *, /)$ de números inteiros são encontrados em [1].

As operações de maior divisor comum de polinômios e divisão de polinômios são encontradas em [4].

3.2.2 – Métodos especializados

3.2.2.1 – Números inteiros e racionais

Ambas as classes possuem em comum métodos referentes à manipulação do sinal dos números e os operadores de comparação ($=$, \neq , $<$, \leq , $>$, \geq).

3.2.2.2 – Números inteiros

A classe de números inteiros possui métodos referentes à paridade do número, aos dígitos mais e menos significativos, à representação em base decimal, hexadecimal ou binária do número e métodos para comparar somente os valores absolutos dos números.

Além disso possui um método para encontrar a raiz quadrada inteira do número

($\lfloor \sqrt{n} \rfloor$) e o resto da divisão inteira de dois números.

3.2.2.3 – Números racionais

A classe de números racionais possui métodos referentes ao numerador e ao denominador e métodos para o inverso multiplicativo do número.

3.2.2.4 – Números em aritmética modular

A classe para números em aritmética modular possui métodos referentes aos dígitos mais e menos significativos do número e referentes à base da aritmética. Bem como métodos para avaliar a congruência e existência de inverso multiplicativo na aritmética.

3.2.2.5 – Monômios e polinômios

As classes de monômios e polinômios têm em comum métodos para: avaliar o objeto em algum valor, remover alguma variável do objeto, obter todas as variáveis ou a variável de maior grau, obter o maior grau de uma determinada variável, obter a primeira derivada parcial em relação a uma variável, obter o conteúdo [4] do objeto, métodos relacionados ao sinal, métodos que dizem se o objeto é um número ou uma variável e métodos que dizem se o objeto é univariado ou multivariado.

Também possuem os operadores de comparação de igualdade e desigualdade.

3.2.2.6 – Monômios

A classe de monômios possui métodos referentes à similaridade de monômios, referentes ao coeficiente do monômio e expoente de uma determinada variável.

3.2.2.7 – Polinômios

A classe de polinômios possuem métodos para se obter o coeficiente líder a partir de uma variável, obter o monômio líder a partir de uma variável, obter o polinômio mônico de maior grau que divide o objeto, obter o número de termos no polinômio e obter a parte primitiva do objeto [4].

4 – Considerações Finais

O objetivo deste projeto foi construir uma biblioteca para matemática simbólica em C++ que seguisse os padrões da *GNU Build System*, bem como explorar diferentes tipos de representação de objetos matemáticos em C++.

A decisão de projeto tomada inicialmente foi de construir o projeto de baixo para cima, começando com uma classe para números inteiros, seguido de classes para números racionais e aritmética modular, então monômios e polinômios e, finalmente, expressões aritméticas.

O projeto poderia ser iniciado em qualquer etapa utilizando outras bibliotecas já existentes para as partes anteriores. Entretanto começar “do começo” é interessante porque o código gerado, menor e mais legível do que em uma biblioteca comercial, pode ser facilmente refatorado para aplicações didáticas. Além de fornecer uma espécie de aquecimento para as etapas posteriores. Foi notável que a experiência obtida com etapas anteriores facilitou cada etapa posterior até o estado atual do projeto.

O projeto ainda tem muito para crescer e evoluir, inicialmente algumas coisas interessantes de se adicionar seriam:

- Mais métodos para manipular expressões e métodos mais inteligentes para simplificá-las
- Implementação de derivação e integração simbólica para funções reais
- Implementação de uma classe para números de ponto flutuante
- Algumas generalizações para a classe de polinômios, como a forma de ordenação dos monômios e os diferentes métodos de divisão
- Classes para vetores e matrizes
- Habilidade de se definir e avaliar regras para as funções

No estado atual, o projeto pode ser útil para aplicações didáticas, pois o código gerado é simples o suficiente para ser manipulado por alunos de graduação e ainda há vários aspectos em que pode ser melhorado (tanto em legibilidade quanto em performance).

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.

APÊNDICE A – Site com código fonte, instruções de instalação e exemplos de uso da biblioteca

<https://www.github.com/ArthurCarmo/mini-cas>

ANEXO A – GNU BUILD SYSTEM

http://www.gnu.org/software/automake/manual/html_node/GNU-Build-System.html