Kíssia Carvalho

Computação Algébrica: Sistemas de Software, Estrutura e Algoritmo de Risch

Dissertação submetida à Coordenação Curso de Pós-Gradução em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, como requisito parcial para obtenção do grau de Mestre em Informática.

Mário Toyotaro Hattori
Orientador

Bruno Correia da Nóbrega Queiroz
Co-orientador

Campina Grande, Paraíba, Brasil

C331c Carvalho, Kissia
Computacao algebrica : sistemas de software, estrutura e Algoritmo de Risch / Kissia Carvalho. - Campina Grande, 1996.
117 f. : il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Sistemas de Software Algebrico 2. Computacao Algebrica 3. Operacoes Aritmeticas 4. Algoritmo de Risch 5. Dissertacao I. Hattori, Mario Toyotaro, M.Sc. II. Queiroz, Bruno Correia da Nobrega, M.Sc. III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 004.41(043)

PARECER FINAL DO JULGAMENTO DA DISSERTAÇÃO DA MESTRANDA

KÍSSIA CARVALHO

TÍTULO: "COMPUTAÇÃO ALGÉBRICA: SISTEMAS DE SOFTWARE, ESTRUTURA E ALGORITMO DE RISCH".

COMISSÃO EXAMINADORA	**CONCEITO**
PROF. MÁRIO TOYOTARO HATTORI, M.Sc - Presidente -	Aprovada
PROF. BRUNO CORRÊIA DA NÓBREGA QUEIROZ, M.Sc - Examinador -	Aprovada
PROF. CLAUDIANOR DE OLIVEIRA ALVES, Dr. - Examinador -	Aprovada
PROF. MAURO CAVALCANTE PEQUENO, Dr.. - Examinador -	Aprovada

Campina Grande, 12 de julho de 1996

Computação Algébrica: Sistemas de Software, Estrutura e Algoritmo de Risch

Kíssia Carvalho

Dissertação de Mestrado aprovada em 12/07 /1996

Mário Toyotaro Hattori
Orientador

Bruno Correia da Nóbrega Queiroz
Co-orientador

Mauro Cavalcante Pequeno
Componente da Banca

Claudianor Oliveira Alves
Componente da Banca

Campina Grande, Paraíba, Brasil, Julho/1996

"Estou certo de que minha mãe virá visitar-me e dar-me os seus conselhos, revelando-me o que nos espera na vida futura"

Santo Agostinho

Ao meus pais Hélio e Rita.

Agradecimentos

Agradeço inicialmente a Deus que me possibilitou reencarnar para que eu buscasse a simplicidade de coração e a humildade de espírito.

Agradeço aos meus pais Hélio e Rita, irmãs Kenia e Kyvânia, meus cunhados, sobrinhos e minha avó Percila.

A Romero, pela sua compreensão e companhia em meus momentos difíceis, e sua família.

As minhas grandes amigas (em ordem alfabética) Joseana, Kátia e Magna pelo apoio durante o meu dia a dia.

A minha primeira professora "Tia Marilita", com quem aprendi a gostar de estudar.

Agradeço a Mendes, Ernesto, José Luiz, Marcos Aurélio e especialmente ao professor Bráulio que me iniciou nos caminhos da álgebra, professores do Departamento de Matemática e Estatística da UFPb.

Aos meus colegas de curso Adna, Aldenor, Carlos, Edjander, Eliane, Gilson, Reginaldo, Robson, Victor Hugo e Washington, cuja ajuda foi de fundamental importância na conclusão deste trabalho.

Aos professores, principalmente meus orientadores Mário T. Hattori e Bruno Queiroz, e aos funcionários, especialmente a Aninha, que fazem a COPIN.

Ao professor da UFC, Mauro Pequeno.

A UFPb e ao CNPq.

E finalmente a mais linda das campinas, Campina Grande.

Resumo

Os problemas de matemática aplicada, em particular os ligados à modelagem, são em geral apresentados em palavras, sem formulas. Isso siguinifica fazer/escolher as hipóteses físicas a serem levadas em conta, ou seja, verificar quais são os igridientes relevantes do problema. Atualmente, pode-se usar o computador, como meio eficiente para experimentação e verificação no estudo da solução destes problemas. A tendencia atual é procurar construir ambiéntes integrados para a solução de problemas (ASP 's) de matemática, o que exige a coperação de muitos especialistas. Há um grande interesse na busca da integração da computação algébrica, computação numérica e computação gráfica, o que conduz a idéia de conceber um sistema de software algébrico organizado em módulos.

Para evitar os erros de arredondamento, os sistemas de software algébricos executam a operações aritméticas básicas por software. Por isso é necessário escolher estruturas de dados adquadas para representar os objetos (inteiros e polinômios) bem como algoritmos que executem essas operações eficientimente. Os números reais podem ser representados de forma a aproveitar os algoritmos e estrutura de dados utilizada para inteiros.

Um sistema de software algébrico deve oferecer recursos de simplificação e recursos para obtenção de primitiva de uma função. Constantes progressos são obtidos, no campo da integração e na busca do desenvolvimento de um "sistema expert" em integração, combinando conhecimentos de computação numérica, computação algébrica e inteligência artificial. O algoritmo de Risch contribuiu significativamente na busca da solução do problema de obtenção de uma primitiva em sua forma fechada. Neste trabalho o algoritmo de Risch é analisado afim de mostrar e compreender detalhes do seu funcionamento.

Na verdade, este trabalho tem a visão moderna segundo a qual, se pode/deve motivar a teoria recorendo-se a situações práticas, fazendo uso do computador. Procurou-se organizar conhecimentos básicos de Computação algébrica para servir de orientação a quem deseja implementar um sistema de computação algébrica ou queira utilizar racionalmente algum sistema existe.

PALAVRAS CHAVES: Sistemas de Software Algébrico, Computação Algébrica, Operações aritméticas, Algoritmo de Risch.

Abstract

Aplicad mathematics problems, specificaly related to modeling, are generally presented in words, without formulas. This means to do/to choose the physical hypotheses to be taken in account or to verify which are the problem main ingredients. Nowadays the computer can be used as an efficient way for experimenting and verification in the study of these problems. The current tendency is looking for building integrated enviroments for the solution of mathematics problems (ASP 's) what demands many experts cooperation. There is a great interest in searching the integration of algebric, numerical and graphic computation that leads the idea of conceiving and algebraic software sistem organized in modules.

To avoid rounding up errors, the algebraic software sistems perform the basic arithmetical operations by software. Is necessary then, to choose data structures apropriated to represent the objects (wholes and polinomes) as well as algoritthms that perform these operations efficiently. Real numbers can be represented in order to utilize the algorithms and data structures used for wholes numbers.

An algebraic software sistem must offer recourses of simplification and recousers to obtain the primitive of a function. A continuous progress is obtained in the field of interaction and in searching to develop an ' 'expert sistem " in interaction, combining numerical computation knowledge, algebraic computation and artificial intelligence. Risch algorithm contributed significantly to find out a solution for the problem of getting a primitive in its closed pattern. In this work Risch algorithm is analysed in order to show and understand details in this functioning.

In reality, this work has a modern view in which you can/must motivate the theory using practical situations applying the computer. It was tried to organize basic knowledges of algebraic computation to serve as orientation to whom desires to implement an algebraic computation sistem or wants to use rationaly some existent sistem.

KEY WORDS: Algebraic Software Sistems, Algebraic Computation, Arithmetical Operations, Risch Algorithm.

Sumário

Lista de Quadros

Lista de Figuras

Capítulo 1

Computação Algébrica

1.1 Introdução

Ao tentar explicar fenômenos da natureza procurando uma relação entre causas e efeitos surgem os problemas de matemática. Busca-se uma explicação para o fenômeno para poder prever o comportamento futuro desse fenômeno. O fenômeno é modelado matematicamente, estabelecendo uma relação entre as possíveis causas e efeitos. Nessa modelagem surgem os problemas de matemática. Dado um problema de matemática, no sentido amplo, para encontrar sua solução existem diversos métodos [1]:

1. Método dedutivo, quando se trata de provar alguma verdade matemática.

2. Método analítico, que procura obter uma relação analítica entre as variáveis independentes e variáveis dependentes satisfazendo alguma(s) condição(ções).

3. Método numérico, que procura obter valores da solução (variável dependente) de um problema para valores discretos das variáveis independentes, quando uma solução procurada for uma função ou os valores numéricos de variáveis que satisfazem alguma condição.

4. Método gráfico, para exibir graficamente o comportamento de um problema ou mesmo de sua solução, cuja inspeção pode fornecer informações importantes sobre o problema e sua solução.

Com o desenvolvimento da tecnologia de informática (hardware e software), atualmente pode-se usar o computador na aplicação dos métodos 2 a 4. O método 1 pode se beneficiar da utilização do computador porque aplicando os métodos 2 a 4 pode-se obter um **discernimento** (*insight*) sobre o problema e este pode sugerir caminhos para se chegar a uma prova.

1.2 Terminologia

Jonh Rice cunhou o termo software matemático (ver definição 1.2.1) em 1969, ao convidar pesquisadores para participarem de um simpósio na Purdue University em abril de 1970[2]. O software matemático teve um desenvolvimento extraordinário na década de 70. Já no fim da década de 70 a comunidade de software matemático começou a tomar consciência de que se preocupou demasiado em criar produtos e se preucupou de menos com os usuários destes produtos.

Com o advento de novos recursos gráficos de hardware e o barateamento das máquinas de uma forma geral, a comunidade de software matemático reavaliou todo o esforço despendido no desenvolvimento de software numérico (ver definição 1.2.2) e chegou à conclusão de que o computador poderia desempenhar um papel muito mais abrangente do que o desempenhado até então, na solução de problemas de matemática. Hoje o software matemático é considerado como uma disciplina (assunto para ensino e pesquisa).

A terminologia usada é introduzida a seguir:

Definição 1.2.1 *[1]: Software matemático é um ramo da engenharia de software que concebe, implementa, testa e mantém software para solução de problemas de matemática.* □

Definição 1.2.2 *[1]: Software numérico é um software cuja finalidade é a solução numérica de problemas de matemática.*□

Definição 1.2.3 *[1]: Software algébrico é um software cuja finalidade é a obtenção de formulas matemáticas como solução analítica de um problema de matemática ou resultado de qualquer manipulação algébrica.* □

Nada impede que um software algébrico obtenha resultados numéricos. Porém, ao contrário do software numérico, um resultado numérico é encontrado obtendo primeiro uma solução analítica que é avaliada a seguir. Um software que seja ao mesmo tempo numérico e algébrico poderia ser qualificado de matemático, o que de fato ocorre na literatura.

Um software combina diversos algoritmos para que cumpra suas finalidades.

Definição 1.2.4 *[1]: Algoritmo básico é aquele definido matematicamente.*□

Definição 1.2.5 *[1]: Algoritmo computacional (numérico, algébrico, ou gráfico) é aquele obtido de um algoritmo básico ou de uma combinação deles, levando em conta as características do computador com a finalidade de executar uma seqüencia de operações para resolver um problema.*□

Definição 1.2.6 *[1]: Computação numérica é a resolução numérica de problemas de matemática utilizando o computador.*□

Definição 1.2.7 *[1]: Computação algébrica ou Manipulação Simbólica é a resolução analítica de problemas de matemática utilizando o computador.*□

Definição 1.2.8 *[1]: Computação gráfica é a geração de gráficos pelo computador e o estudo de algoritmos, técnicas e métodos para essa geração.*□

O software matemático precisa combinar os conhecimentos de ciência da computação, da computação numérica e da engenharia, sem mencionar os da matemática que permeia toda as três áreas.

Um software matemático pode se apresentar como:

- Um pacote: coleção de programas para resolver uma classe ou uma grande subclasse de uma classe de problemas;
- Uma biblioteca: coleção de programas para resolver diversas classes de programas de matemática;
- Um sistemas de software: constituído de um pacote ou de uma biblioteca mais uma interface do usuário.

Um software para computação algébrica em geral se enquadra na classificação de um sistema de software. É composto de um elenco de recursos e de uma linguagem para manipular esses recursos. Por conveniência um sistema de software para computação algébrica, será chamado aqui como sistema algébrico ou sistema de software algébrico.

1.3 Computação Algébrica

Hamming [3] afirma que "A finalidade da computação é o discernimento, não números". O discernimento raramente é obtido a partir de uma grande quantidade de números; graficamente é a forma mais comum de obtê-lo. Existem outras maneiras de se obter o discernimento – o uso da manipulação simbólica pelo computador (também conhecida como computação algébrica).

Na verdade, a idéia de que qualquer procedimento analítico exato poderia ser executado por um computador foi expressa em 1844 por Lady Lovelace (Ada Augusta), a filha de Lord Byron e patronesse do matemático inglês Charles Babbage. Em 1833 Babbage desenvolveu uma "máquina analítica", que era uma espécie de máquina de calcular progamável com aritmética e dispositivos de memória. Cem anos mais tarde, as propriedades da máquina de Babbage tornaram-se os fundamentos dos computadores modernos [4]. A primeira tentativa bem sucedida para executar uma operação não numérica simples, em um computador ocorreu em 1955: a diferenciação [5, 6]. O próximo e importante passo nessa direção – o desenvolvimento da álgebra polinomial – necessitou de outros 10 anos de desenvolvimento de tecnologia de hardware, de novos algoritmos [7], e a criação de linguagens de programação.

Nos anos 60 a idéia de manipulação algébrica, que usa variáveis ao invés de números, marcou o nascimento de uma área de especialização, chamado *Computação Algébrica* ou *Manipulação Simbólica*. Observou-se que um computador pode auxiliar um cientista ou engenheiro, não só em operações com números, mas também analisando o comportamento desses problemas.

A manipulação simbólica pode ajudar a resolver problemas de três maneiras [8]:

- fornecendo uma resposta completa;
- fornecendo uma aproximação simbólica;
- auxiliando na previsão de uma resposta numérica;

Muitas pessoas se supreendem ao saber que os computadores são capazes de executar manipulação algébrica tão bem quanto operações numéricas. A surpresa ocorre porque geralmente o computador é visto como uma ferramenta destinada para computação numérica.

Na computação numérica, os símbolos que são usados representam números, e os resultados obtidos também são números. Com sistemas algébricos, por outro lado, é possível usar computadores para executar analiticamente operações como diferenciação, simplificação de expressões (coletar termos similares), substituição de símbolos ou expressões por outras expressões, etc.

Um programa de computação algébrica apresenta algumas vantagens em relação aos programas de computação numérica. A primeira delas é a economia de tempo de computação, quando uma expressão algébrica for simplificada antes de ser avaliada numericamente. A segunda vantagem é a obtenção de soluções, quase sempre exatas, em contraste com as soluções numéricas que são *a priori* aproximadas. Um exemplo prático é a diferenciação númerica que tem muitas desvantagens comparada com a diferenciação simbólica [9]. Note-se que um resultado na forma algébrica pode facilitar sua análise e interpretação.

Vários algoritmos foram concebidos para diferenciação, integração e expansão de expressões algébricas. A Computação Algébrica se desenvolve para fazer com que os recursos do computador se tornem disponíveis e acessíveis para o mundo educacional, científico e industrial [10], executando as tarefas necessárias da matemática com mais eficiência ou que eram impossíveis manualmente. Um software de computação algébrica é uma ferramenta que permite ao usuário ter acesso aos conhecimentos de matemática de forma natural, e a cada dia de forma mais intuitiva, devido à evolução das interfaces entre usuário e computador.

Como o computador é programado para realizar as manipulações algébricas? A idéia é fazer com que o computador siga um rigoroso procedimento que especifique todos os passos a serem seguidos. No desenvolvimento de um algoritmo para computação algébrica nem sempre se segue o procedimento que seja o mais eficiente manualmente. Procura-se construir um algoritmo que represente uma expressão da forma mais simples possível. Entretanto, pessoas discordam sobre o que constitui a forma mais simples de uma expressão, porque a utilidade de uma forma particular depende das circunstâncias em que o usuário se encontra. Por isso, a construção de um algoritmo geral para simplificação é impossível.

É importante observar que sistemas algébricos não devem ser empregados apenas

em cálculos complexos, isto é, cálculos que envolvem expressões com muitos termos e várias funções matemáticas. Problemas relativamente simples abordados por manipulação algébrica podem contribuir para tornar a avaliação numérica mais simples.

1.4 Aplicações

A computação algébrica tem aplicações em várias áreas: economia, geometria algébrica, mecânica dos fluidos, física, engenharia, etc [11, 12], cujos especialistas dessas áreas devem contar com uma ferramenta adequada.

Um exemplo clássico do auxílio da computação algébrica ocorre na teoria do movimento da lua, que fornece a posição da lua como função do tempo. A solução foi obtida manualmente em 1867, pelo astrônomo francês Charles Delauney, que levou 10 anos para sua conclusão e mais 10 para verificá-lo. Em 1970 três erros foram detectados após 20 horas de processamento, por um programa escrito por André Deprit, Jacques Henrard e Arnold Rom, cientistas dos Boeing Scientific Research Laboratories em Seattle. Apesar da computação algébrica ter sido responsável pela exposição do erro de Delaunuay, a tabela por ele construída, que relaciona a posição da lua com o tempo, tem servido de parâmetro para os novos sistemas de computação algébrica [11].

A Computação algébrica pode ser aplicada na computação numérica para diminuir o tempo de processamento com a simplificação algébrica de uma expressão antes de ser avaliada numericamente. O exemplo a seguir ilustra a aplicação de métodos de solução de problemas de matemática.

Para encontrar a soma dos 100 primeiros inteiros numericamente usam-se 198 adições separadas (ver Figura 1.1). Um algoritmo empregado por um programa de computação algébrica resolve o problema considerando que a soma dos n primeiros inteiros é um polinômio do segundo grau em n. Da mesma forma, o algoritmo pode encontrar a soma dos quadrados dos n primeiros inteiros, construindo um polinômio do terceiro grau como função de n. Apesar desse algoritmo ser eficaz, não é a forma mais rápida de resolver o problema de somar os n primeiros inteiros. Existe, por exemplo, o método proposto por Gauss, que percebeu que os inteiros de 1 a 100 poderiam ser agrupados em 50 pares cuja soma é igual a 101, restando apenas fazer o produto 101 x 50, mas é um método que não serve para encontrar a soma de quadrados, cubos ou altas potências de inteiros enquanto que o algoritmo de computação algébrica pode.

Pela bibliografia, verifica-se que o campo de aplicação da computação algébrica é

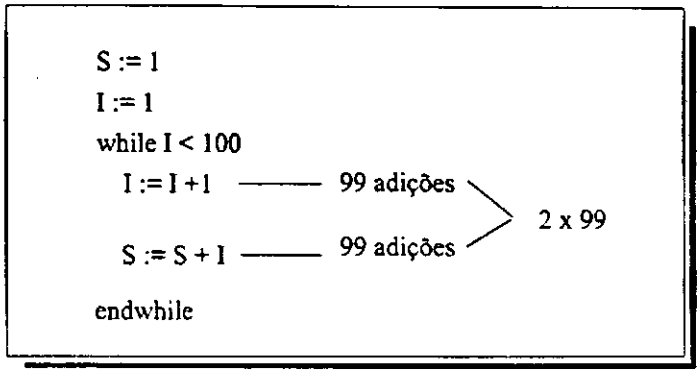

Figura 1.1: Algoritmo para o cálculo da soma dos 100 primeiros inteiros

amplo. Apesar da computação algébrica ter conseguido resolver problemas em muitas aplicações importantes, não se pode dizer que as pesquisas na área estejam esgotadas.

1.5 Objetivos da Dissertação

A dissertação procura organizar conhecimentos básicos de Computação Algébrica para servir de orientação a quem deseja implementar um sistema de computação algébrica ou queira utilizar racionalmente algum sistema existente.

Estuda os algoritmos para computação algébrica e tenta classificá-los em grupos, especificando:

- Que grupos de algoritmos são indispensáveis em qualquer sistema de manipulação algébrica.
- Quais os algoritmos que realizam as operações aritméticas básicas.
- Estuda o algoritmo de Risch (para obtenção de primitivas) e faz algumas considerações sobre a maneira de colocá-lo de uma forma que seja possível implementá-lo.

1.6 Organização da Dissertação

A dissertação foi assim organizada:

- *Capítulo 2 - Sistemas de Software para computação Algébrica.* Levantamento histórico, classificação, propriedades, aplicações de sistemas algébricos, enfatizando a simplificação algébrica. Tendências atuais da integração da computação numérica, computação algébrica e computação gráfica são também discutidas.

- *Capítulo 3 - Organização de um Sistema de Software Algébrico.* Apresenta uma arquitetura de um Sistema de Software Algébrico, em que são identificadas as operações necessárias para a construção de módulos de um sistema, além de uma breve análise sobre os atributos da interface do usuário e da linguagem a ser utilizada por um sistema.

- *Capítulo 4 - Operações Aritméticas Básicas.* São apresentados e discutidos os algoritmos para execução de operações aritméticas básicas com inteiros e polinômios em precisão arbitrária. É também discutida a manipulação dos números reais.

- *Capítulo 5 - Algoritmo de Risch.* Uma breve descrição sobre integração simbólica. O algoritmo de Risch (para encontrar primitivas) é analisado a fim de mostrar e compreender os detalhes do seu funcionamento, colocando-o de uma forma que seja possível implementá-lo. Além disso, é apresentado um breve estudo comparativo de como alguns sistemas algébricos que tratam do problema de integração.

- *Capítulo 6 - Conclusões.* São apresentadas algumas conclusões e sugestões para trabalhos futuros.

Capítulo 2

Sistemas de Software para Computação Algébrica

Desde meados da década de 60 mais de 30 sistemas foram desenvolvidos. Os sistemas de software algébrico podem simplificar expressões que são grandes (contém muitos termos) e complexas. A simplificação é a mais interessante e controvertida operação na manipulação algébrica, e é importante saber que essa operação troca a forma e não o valor de uma expressão.

Este capítulo apresentará um levantamento histórico, uma classificação, propriedades, aplicações de sistemas de software algébrico enfatizando a simplificação algébrica.

2.1 Conceitos Básicos

Na década de 60, houve duas abordagens na construção de sistemas algébricos, que geraram sistemas que podem ser classificados em dois grandes grupos [1, 13]:

- Sistemas com finalidade geral desenvolvidos para executar manipulação algébrica de modo geral. Nessa abordagem foram descobertas as limitações do computador para realizar as manipulações e indentificada a maioria dos problemas de implementação. Os cientistas engajados nessa abordagem procuraram resolver esses problemas.

- Sistemas destinados à solução de problemas específicos, especializados em resolver classes de problemas como os de física teórica, astronomia e química. Os cientistas que adotaram essa abordagem estavam interessados em resolver os seus problemas e não o problema mais geral de computação algébrica. Nesse grupo a computação algébrica foi considerada como uma ferramenta, ao contrário do grupo anterior que tinha como objetivo resolver os problemas de computação algébrica sem se preocupar com problemas específicos.

Exemplos do primeiro grupo são o REDUCE que tem versão até para PC, MACSYMA que tem versão para VAX chamado VAXIMA e SCRATCHPAD desenvolvido pela IBM. No segundo grupo os exemplos mais conhecidos são ASHMEDAI para eletrodinâmica quântica, CAMAL para teoria lunar e da relatividade geral, SCOONCHIP para física de alta energia, SHIP para manipulação de tensores e TRIGMAN para mecânica celeste.

Os dois grupos aprenderam um com o outro e continuam aprendendo. Cada um desenvolveu seus sistemas; atualmente não existem sistemas algébricos que tenham sucesso em todas as áreas de especialização; por isso o aprendizado dos grupos é contínuo.

Os sistemas algébricos podem ser classificados em três grupos principais que refletem o desenvolvimento histórico [11]. Os sistemas do primeiro grupo são descendentes dos sistemas concebidos no início da computação algébrica: eles foram concebidos para resolver problemas específicos em determinadas áreas da matemática, física e química. Os sistemas do segundo grupo são de finalidade geral, incluindo as quatro operações aritméticas, cálculo de integrais definidas e indefinidas, resolução de equações, inclusive equações diferenciais ordinárias, expansão de uma função em série de Taylor, etc. A última geração tem a adição de recursos gráficos e se preocupa com a comunicação entre o usuário e o computador. Essa geração foi viabilizada pelo barateamento e avanço da tecnologia de hardware.

Por inúmeras razões leva-se um longo tempo para o desenvolvimento de programas de manipulação de expressões algébricas. A dificuldade mais óbvia encontrada é a inadequação do hardware do computador para manipulação de expressões algébricas e, conseqüentemente, operações fáceis com adição e multiplicação devem ser executadas por programa.

O controle de expressões intermediárias [14], conhecido como "explosão", é outro problema. Tobey descreve o problema como "o inchaço da expressão intermediária"

[15]. Quando uma expressão algébrica for escrita em um papel não será sempre do mesmo tamanho. Da mesma forma, quando as expressões forem representadas na memória do computador não ocuparão o mesmo espaço. Qualquer caso simples envolvendo manipulação de expressões, começa e termina com expressões simples, mas freqüentemente envolve expressões intermediárias de complexidade arbritrária. Se a memória não puder ser alocada para expressões intermediárias, a execução do programa não poderá ser concluída. Por exemplo, um autovalor de uma matriz cujos elementos são polinômios pode ser um número. Entretanto, a fim de obtê-lo, é preciso obter analiticamente o determinante dessa matriz que pode ser uma expressão muito complicada.

Na computação numérica existe uma forma racional de distribuir os dados na memória e prever a possível demanda de espaço pelos dados de saída, antes de iniciar a execução do programa. Para o manipulação algébrica às vezes é impossível alocar antecipadamente memória para armazenar expressões, ou determinar quanta memória será necessária em passos intermediários. Uma forma de resolver esse problema é usar alocação dinâmica de memória nos passos intermediários do processamento. Os dados que não forem necessários adiante, são descartados, e a memória torna a ser usada para armazenar novos dados.

A distinção principal entre a manipulação simbólica e a computação numérica está na grande complexidade das operações elementares (adição, multiplicação, etc.). Na manipulação simbólica as operações elementares são normalmente executadas por programa, fazendo com que as instruções necessárias para um controle de entrada e saída dos dados sejam mais complicadas.

Em alguns casos a manipulação simbólica pode ser executada fazendo uso direto de números que são facilmente armazenados na memória do computador em código binário. Por exemplo o polinômio $54/7x^3yz^2$, nas variáveis x, y, z, w, pode ser representado de uma maneira clara pelo conjunto de números 54, 7, 3, 1, 2, 0. Essa representação é uma forma simples de desenvolver um sistema compacto e rápido, mas quando se trabalha com operações simbólicas como diferenciação o programa apresenta dificuldades.

Quatro métodos foram usados para executar manipulação simbólica no computador [12]:

Primeiro Método - Os dados de entrada são armazenados na memória de forma compacta como foi descrita acima e o programa necessário para a manipulação desses dados são escritos em linguagem de baixo nível. É comumente usado para resolver uma

pequena porção de problemas. É a base para muitos programas de sistemas algébricos que são compactos e simples. Um exemplo é o sistema SCHOONSCHIP.

Segundo Método - Aqui os dados também são armazenados de forma compacta, mas os programas para manipulação desses dados são escritos em linguagem de alto nível ou em forma de subprogramas. Alguns subprogramas são escritos em linguagens de alto nível e outros em linguagem de baixo nível. Esses subprogramas são ativados em uma linguagem de alto nível. Um exemplo é o sistema SAC-I.

Terceiro Método - As operações necessárias são programadas diretamente em linguagens de alto nível. O armazenamento de dados e sua manipulação são executados de arcordo com um conjunto de regras definido por uma linguagem particular. Neste método a representação interna dos dados não é necessariamente compacta.

Quarto Método - É a base para a maioria dos sistemas algébricos sofisticados. Este método envolve o uso de um conjunto de subprogramas de manipulação escritos em qualquer linguagem de manipulação simbólica (por exemplo LISP). O sistema incorpora um grande número de funções especiais, que são implementadas independentemente. Os dados de entrada e o programa de controle são escritos numa linguagem especial, desenvolvida com o resto do sistema. Essa linguagem deve ser a mais conveniente para o usuário, entre outras coisas. Um exemplo é o sistema REDUCE-2 [16].

Observe os quadros 2.1, 2.2, 2.3 e 2.4 [12], que fornecem algumas informações sobre os sistemas citados como exemplos na classificação acima.

Quadro 2.1: Propriedades gerais de sistemas algébricos

Sistemas	SCHOONSCHIP	CLAM	REDUCE-2
versão(ano)	1977	1972	1973
computador	CDC-6500	CDC-6500	CDC-6500
linguagem de implementação	Assembler	Assembler	LISP
linguagem externa	-	LISP	ALGOL
aplicações primárias	QFT	GTR	Universal

Linguagem especiais dos sistemas:

a)QFT (Quantium Field Theory - Teoria Quântica dos Campos);

b)GTR (General Theory of Relativity - Teoria geral da relatividade);

c)CM (Celestial Mechanics - Mecânica Celeste);

Quadro 2.2: Propriedades gerais de sistemas algébricos (continuação)

Sistemas	SYMBAL	CAMAL	MACSYMA
versão(ano)	1970	1975	1977
computador	CDC-6500	EC-1040	DECPDP-10
linguagem de implementação	Assembler	BCPL	LISP
linguagem externa	ALGOL	-	ALGOL
aplicações primárias	GP	CM e GTR	Universal

d)MP (Mathematical and Physics - Matemática e Física);

e)GP (General Pourpose - Finalidade geral).

Suponha que as manipulações algébricas foram concluídas no computador. Agora é preciso que o usuário seja capaz de compreender a resposta, o que constitui um outro tipo de problema: como o resultado da manipulação algébrica pode ser apresentado ao usuário utilizando os periféricos disponíveis no computador? Além disso, o significado de uma expressão depende do contexto em que o usuário se encontra; por exemplo, a expressão v^i, em algumas áreas da matemática significa o i-ésimo componente do vetor v, em outro significa v elevado a potência i. Progressos significativos foram feitos na resolução do problema de ajuste ao contexto, mas esse problema não foi ainda completamente resolvido. Não se espera que as dificuldades de usar sistemas algébricos venham a desaparecer por completo. Os usuários a cada dia vão exigir mais clareza e os engenheiros de software vão tentar aperfeiçoar suas criações.

2.2 Simplificação Algébrica

Os sistemas algébricos podem simplificar expressões que são grandes e complexas. Tais expressões podem conter não apenas funções elementares como trigonométricas, logaritmicas e polinomiais, mas também funções mais complexas envolvendo derivadas e funções especiais como as de Bessel. O usuário do sistema algébrico pode também definir sua função, especificar suas propriedades e impor regras de simplificação apropriadas; na verdade, o computador pode, às vezes, ser empregado para encontrar tais regras.

Simplificação é a mais interessante operação na manipulação simbólica. É também

Quadro 2.3: Objetos matemáticos e operações com eles que foram incorporados nos sistemas do quadro 2.1

Sistemas	SCHOONSCHIP	CLAM	REDUCE-2
Funções elementares	Não	Muitas	Algumas
Funções racionais	Não	Sim	Sim
Máximo divisor comum	Não	Não	Sim
Diferenciação	Não	Sim	Sim
Integração	Não	Não	Não
Números complexos	Sim	Não	Sim
Números racionais	Sim	Sim	Sim
Aritmética de números flutuantes e decimais	Rápido	Não	Lenta
Séries de potência	Não	Não	Moderado
Séries de Fourier	Não	Não	Não
Vetores e tensores	Moderados	Alguns tipos	Bom
Álgebra de matrizes	Excelente	Não	Bom

Quadro 2.4: Objetos matemáticos e operações com esses objetos que foram incorporados nos sistemas do quadro 2.2

Sistemas	SYMBAL	CAMAL	MACSYMA
Funções elementares	Poucas	Muitas	Todas
Funções racionais	Sim	Sim	Sim
Máximo divisor comum	Não	Casos simples	Sim
Diferenciação	Sim	Sim	Sim
Integração	Casos simples	Casos simples	Sim
Números complexos	Sim	Sim	Sim
Números racionais	Sim	Sim	Sim
Aritmética de números flutuantes e decimais	Não	Rápido	Rápido
Séries de potência	Excelente	Bom	Excelente
Séries de Fourier	Tipos especiais	Excelente	Bom
Vetores e tensores	Moderado	Moderado	Excelente
Álgebra de matrizes	Não	Não	Não

a mais controvertida. Entretanto, existe uma propriedade, com a qual usuários e engenheiros de software concordam. A simplificação troca a forma ou a representação de uma expressão mas não o seu valor.

Historicamente, a simplificação foi exigida dos sistemas de manipulação simbólica para que produzisse resultados mais naturais ao usuário. Por exemplo, o resultado da diferenciação de $ax + xe^{x^2}$ em relação a x, de forma não simplificada é

$$0 \cdot x + a \cdot 1 + 1 \cdot e^{x^2} + x \cdot e^{x^2} \cdot 2 \cdot x.$$

Simplificando, a expressão acima se torna:

$$a + e^{x^2} + 2 \cdot x^2 \cdot e^{x^2}.$$

O problema de representação de uma expressão algébrica é um caso especial da manipulação simbólica porque existem muitas formas. Freqüentemente uma dessas formas equivalentes é mais usada que outra e não é um problema trivial reconhecer essa equivalência.

Os projetistas preferem sistemas nos quais os passos da simplificação são os mesmos em qualquer contexto. Usuários preferem um sistema que produza uma expressão simplificada que lhe forneça uma informação contextual.

Alguns usuários estão apenas interessados no resultado final da execução de um programa. Para tais usuários o problema de simplificação se reduz a obter uma expressão de forma a otimizar o uso do espaço de memória e do tempo de computação. Por exemplo, para aqueles que usam a diferenciação simbólica como uma ferramenta para obtenção do valor de uma derivada, só interessa a forma final da função derivada.

Outros usuários, desejam observar algumas propriedades de um fenômeno através da manipulação do modelo matemático, procurando uma interpretação física. Por exemplo, quando ele está interessado na maneira como o valor de uma expressão varia com uma de suas variáveis.

É mais fácil para o usuário compreender e responder questões sobre expressões, com poucos termos do que sobre expressões com muitos termos, o que faz supor que a meta da simplificação é produzir expressões com poucos termos. E, de fato a maioria das simplificações usuais produz expressões com menos termos. Entretanto, há simplificações

que produzem expressões com mais termos, por exemplo; $(x+1)^3 \mapsto x^3+3x^2+3x+1$ e outras produzem expressões com menos termos $x+2x \mapsto 3x$. Muitos sistemas só executam tais transformações se o usuário desejar.

2.3 Classificação dos Sistemas Algébricos de acordo com a Simplificação

A simplificação é de tanta importância em um sistema algébrico, que quando um engenheiro de software decide como vai representar as expressões, quais as mudanças que o sistema poderá fazer automaticamente, quais dessas mudanças vão ser permitidas ao usuário ignorar e modificar, e quais as facilidades adicionais para simplificar expressões que o sistema irá ter, restar-lhe-ão poucas decisões importantes a tomar. A simplificação é tão importante que pode ser usada para classificar sistemas de software algébrico.

Antes de prosseguir com a classificação é razoável que se defina, o que seja a forma canônica de uma expressão. Observa-se que não existe uma definição geral de forma canônica de uma expressão. Cavinness [17] faz considerações sobre a forma canônica de uma expressão com respeito a simplificação algébrica.

Obter uma classe de expressões $\mathcal{E}$ significa obter um conjunto de regras para determinar uma "bem-definida" expressão da classe. As expressões devem ser consideradas como um conjunto de símbolos atômicos, no qual um subconjunto é designado por *variáveis*. Qualquer membro de $\mathcal{E}$ que não contenha uma variável é chamado uma *$\mathcal{E}$-constante*. A expressão passa a ser interpretada como função sobre algum domínio $\mathcal{D}$.

Se E_1 e E_2 forem membros de uma classe de expressões $\mathcal{E}$, E_1 é *idêntica* a E_2 se e somente se E_1 e E_2 possuirem o mesmo *string* de símbolos atômicos. Essa relação é denotada por $E_1 \equiv E_2$. E_1 e E_2 são *equivalentes*, se para todos os valores em $\mathcal{D}$ atribuídos às suas variáveis a mesma resposta para E_1 e E_2 é obtida; denota-se por $E_1 = E_2$.

A definição de forma canônica e forma normal de uma expressão é dada aqui exclusivamente como condições suficientes para sua existência.

Definição 2.3.1 [17]: *Uma forma f-normal para uma classe de expressões $\mathcal{E}$ é uma função f de $\mathcal{E}$ em $\mathcal{E}$ que satisfaz as seguintes propriedades para qualquer E em $\mathcal{E}$.*

(i) $f(E) = E$, *e*

(ii) Se $E = 0$, $f(E) \equiv 0$.□

Definição 2.3.2 [17]: *Uma forma f-canônica é uma forma f-normal com a propriedade adicional de que para todo* E_1 *e* E_2 *em* $\mathcal{E}$ *têm-se* $E_1 = E_2$ *e* $f(E_1) \equiv f(E_2)$.□

Se em particular f for transparente ao contexto ou f for arbitrária, usa-se apenas a forma canônica (normal) ao invés de forma $f-$canônica (normal).

Definição 2.3.3 [17]: *Uma classe de expressões é chamada classe canônica (normal) se existe uma forma canônica (normal) para ela.*□

Para um dado conjunto de expressões $\mathcal{E}$ e uma forma canônica (normal) f para $\mathcal{E}$, os membros do conjunto $f(\mathcal{E})$ de uma expressão canônica vão ter uma certa forma em comum. Por exemplo, uma forma canônica para polinômios com coeficientes racionais mapeia cada polinômio para a forma $r_0 + r_1 x + ... + r_n x^n$, onde os r_i 's são números racionais na forma canônica.

A classificação é baseada no seguinte critério: o nível em que o sistema troca a representação fornecida pelo usuário. Eles são classificados em [13]:

Radicais

Os sistemas radicais podem manipular uma bem definida classe de expressões (por exemplo, polinômios, funções racionais, séries de potências). As expressões nesse caso são representadas na forma canônica, isto é, quaisquer expressões equivalentes são internamente representadas de maneira idêntica. Isso significa que o sistema muda totalmente a expressão fornecida pelo usuário a fim de obter a expressão em sua forma canônica. A vantagem desse sistema está na realização de trabalho baseada em um bem definido e eficiente algoritmo de manipulação. Uma desvantagem está na expansão de expressões como $(x + 1)^{1000}$, cuja obtenção da primitiva ou derivada, por exemplo, é muito mais fácil sem expandi-la. Exemplos de sistemas radicais são SAC-I e o ALPAK [18, 19].

Deixa a nova expressão *(New Left)*

Esses sistemas superam algumas dificuldades causadas pela expansão automática de expressões obtidas ou fornecidas pelos sistemas radicais. O usuário do *new left* pode

especificar quando uma expansão de uma expressão deve ser realizada. Esse sistema pode simplificar uma variedade de expressões com grande facilidade, mas não tão eficientes quanto os sistemas radicais. Exemplos de sistemas *new left* são o REDUCE e o ALTRAN [20, 21].

Os Liberais

Os sistemas liberais contam com uma variedade de representações de expressões e usam regras de simplificação manual. Esses sistemas executam simplificação usual, organizando somas e potências de produtos, aplicando regras em relação a coeficientes 0 e 1 e removendo operadores redundantes.

A grande desvantagem desse sistema está na representação de uma informação que requer de duas ou três vezes mais espaço de memória que num sistema radical e a manipulação torna-se bastante lenta em algumas situações. A vantagem é a capacidade de expressar problemas de maneira mais natural ao usuário que os dois sistemas anteriores. FORMAC [22] é um exemplo de sistema liberal.

Os Conservadores

Os sistemas conservadores procuram não só executar simplificações, mas também se adequar a ocasiões. Os engenheiros de software o conceberam para possibilitar ao usuário construir suas simplificações e mudá-las quando necessário. Esses sistemas são ainda mais lentos que os liberais.

A importância dos sistemas conservadores está na filosofia que eles implementam [23]. A simplificação de uma expressão deve ser determinada pelo contexto. Os engenheiros de software entendem que o sistema deve ser capaz de se ajustar à natureza particular do problema. FAMOUS e FORMULA ALGOL [23, 24] são exemplos de tais sistemas.

Os Caóticos

Os sistemas caóticos usam mais de uma representação para uma expressão e fornece mais de uma abordagem para a simplificação. A idéia básica é admitir que se uma técnica não funciona a outra deve funcionar. O usuário deve ser capaz de escolher qual das simplificações fornecidas é adequada em cada caso. Os projetistas do sistema caótico enfatizam a habilidade de resolver uma grande variedade de problemas. Eles procuraram fornecer ao usuário a eficiência e a força dos sistemas radicais e a atenção ao contexto dos sistemas conservadores. A desvantagem dos sistemas caóticos é a sua organização e seu tamanho. Eles são necessariamente maiores que os outros sistemas.

Um exemplo desse tipo de sistema é o SCRATCHPAD [25], atualmente conhecido como AXIOM, que tem quatro simplificadores.

2.4 Aplicações dos Sistemas Algébricos

Existem mais de 500 publicações sobre a utilização de sistemas algébricos em física e matemática [12]. Os sistemas algébricos, no entanto, podem ser usados nas áreas mais inusitadas, já que antes de tudo os projetistas dos sistemas algébricos têm por objetivo fazer com que o usuário obtenha **discernimento** sobre o problema de matemática estudado. Em hidrodinâmica e em muitos outras áreas da matemática aplicada é necessário resolver sistemas complicados de equações diferenciais parciais. Os sistemas algébricos podem ajudar a resolver essas equações [12].

O algoritmo de Euclides, é um método sistemático para encontrar o máximo divisor comum de dois elementos de um domínio euclidiano (ver definição no Apêndice). É possível que na aplição desse método haja um crescimento dos resultados intermediários. Por essa razão os objetos, ou seja, os elementos do domínio euclidiano, devem ser armazenados no computador, em uma estrutura dinâmica que permita que esses objetos cresam arbitrariamente. Geralmente um número armazenado no computador usa um espaço fixo na memória, mas em um sistema de manipulação algébrica a alocação fixa não pode ser usada. Matemáticos que estudam as propriedades dos grandes números foram seduzidos por essa característica dos sistemas algébricos [11].

O matemático George David Brikhoff da Universidade de Harvard demonstrou, em 1923, um teorema declarando que a teoria geral da relatividade exclui a propagação no espaço de pulsos gravitacionais cuja energia conceitualmente é gerada pela pulsação radial da matéria na estrela. Devido a nenhum pulso ter sido detectado e devido ao resultado de Brikhoff contrariar as teorias de Newton e de Einstein, a exclusão de pulsos passou a ser desconsiderada em qualquer teoria gravitacional. Era cansativo e lento determinar quando a teoria da gravidade satisfaz ou não o teorema de Birkhoff; a teoria gravitacional elaborada por C. N. Yang da Universidade de New York em Stony Brook mostrou a utilidade do sistema algébrico para testar a validade do teorema de Birkhoff [14].

2.5 A Tendência Atual de Integração da Computação Algébrica, Computação Numérica e Computação Gráfica

Como foi dito no capítulo 1, há métodos diferentes de atacar a solução de um problema de matemática. Em geral, esses métodos não são usados isoladamente em ciência e tecnologia.

Em meados da década de 1980 [26, 27] começaram os esforços para acoplar processamento numérico com simbólico (técnicas de inteligência artificial). Esses esforços têm por objetivo aperfeiçoar os mecanismos de adaptabilidade presentes em software de equações diferenciais ordinárias e software de quadratura inventados na década de 1970. O que se procura é acompanhar criticamente a execução de um algoritmo numérico e, dependendo da solução intermediária ou parcial, alterar um ou mais parâmetros do algoritmo ou ainda mudar de algoritmo.

Os especialistas chegram à conclusão de que um computador poderia auxiliar um cientista ou engenheiro não só com cálculos numéricos, manipulando formulas ou exibindo graficamente o comportamento de um problema, mas orientando-o na execução de suas tarefas, ou seja, poderia embutir alguma "inteligência" de modo que o comportamento do sistema dependesse das decisões do usuário [1]. A idéia por trás de um ambiente de solução de problemas é de que os sistemas que ajudam a resolver problemas de matemática devem integrar os recursos de computação numérica, computação algébrica e computação gráfica e além de incorporar subsistemas de inferência e de banco de dados diversos. Além disso, a eliminação da tarefa de programar a solução de um problema de matemática numa linguagem tradicional é altamente desejável.

O conceito de um ambiente de solução de problemas (ASP) de matemática ainda não está totalmente estabelecido. Sabe-se razoavelmente o que se quer que um ASP faça, mas ainda não está claro como integrar num sistema as tecnologias de computação numérica, computação algébrica e computação gráfica com as tecnologias de inteligência artificial, banco de dados e de linguagens de programação. Questões como aproveitar o software existente nas diversas áreas ou inventar uma nova linguagem de programação para viabilizar o desenvolvimento de ASP 's ainda precisam ser respondidas.

Para que um sistema seja considerado um ASP:

- deve apresentar uma aparência integrada e unificada ao usuário:

- deve conter um subsistema de raciocínio automático;
- as diversas ferramentas que o compõem devem se comunicar de forma clara ou mesmo transparente ao usuário; e
- deve se apresentar ao usuário com a estrutura de um sistema especialista.

As vantagens que um ASP pode trazer são as seguintes:

- reflete de modo natural a metodologia de modelagem usada em ciência e tecnologia;
- serve de arcabouço para pesquisa científica;
- assiste um pesquisador em sua experimentação; e
- amplia os conhecimentos de uma área e ao mesmo tempo permite detetar falhas na maneira de pensar.

Se há vantagens em se construir e utilizar ASP 's há também desvantagens:

- Não se sabe como construí-los e mesmo como defini-los adequadamente.
- O tempo necessário para descobrir como criá-los e como usá-los é provável que seja substancial e cada ASP poderá ser arbitrariamente diferente de outro e não ser portátil (compilável e executável em um grande número de ambientes de computação sem nenhuma alteração).
- Desenvolver software colaborativo (em dois sentidos: um software que ajuda outro e participação de vários especialistas) de alta qualidade é muito difícil.
- O sucesso no desenvolvimento e utilização de ASP's dependerá da disponibilidade de informações de qualidade e de documentação para que todas as partes do ASP sejam usadas.

Os usuários de ASP's serão cientistas, engenheiros, matemáticos, estudantes e outros. Diferentes usuários procurando acesso a um ASP o desejarão por diferentes razões. Há contudo, certas características que todos esperam de um ASP. Tais características

se referem principalmente aos aspectos de assistência, gerência, manutenção e confiabilidade. Esses aspectos constituirão um fator relevante na adoção ampla e na utilização de um ASP.

A matemática necessária deve atender a ajustamento de curvas e superfícies com exibição gráfica, estatística e manipulação simbólica. Algoritmos numéricos constituem requisitos gerais em computação usada em ciência e tecnologia.

Um aspecto deve distinguir um ASP de outros programas de aplicação. Deve ser uma ferramenta para ajudar a resolver problemas novos, respondendo a questões dos usuários que normalmente são difíceis e não rotineiras. Questões de eficiência do sistema são importantes, mas freqüentemente subordinadas à otimização do esforço e tempo gastos pelo usuário.

Capítulo 3

Organização de um Sistema de Software Algébrico

No capítulo 2 foram apresentados um levantamento histórico, uma classificação, propriedades e aplicações de sistemas algébricos, enfatizando a simplificação algébrica. Destacou-se também a tendência atual em integrar a computação algébrica, computação numérica e computação gráfica. Neste capítulo será apresentado um projeto de arquitetura de um sistema de software algébrico, considerando a possível integração das três áreas da computação. São identificados os algoritmos que são indispensáveis em qualquer sistema algébrico, agrupando-os em módulos que permitem alcançar maior clareza na estrutura lógica do software. Também são feitas considerações sobre a interface do usuário e a linguagem de um sistema de software algébrico.

3.1 Modelo de Arquitetura de um Sistema de Software Algébrico

Os programas de computação algébrica são ferramentas para solução de problemas de matemática. Os passos envolvidos na construção de um sistema algébrico (definido no capítulo 1) podem ser descritos em termos gerais. Todo sistema tem características especiais para atender os seus usuários. À medida que crescem o tamanho e a complexidade da aplicação, as considerações sobre o procedimento para construção do software tornam-se mais importantes.

Em uma visão geral, pode-se dizer que os sistemas são comumente divididos em (ver Figura 3.1):

- núcleo, onde estão as rotinas que controlam a operação do sistema e executam as operações básicas e as aritméticas;
- a interface, que controla a comunicação com o usuário; e
- banco de funções ou módulos que contém procedimentos para execução de operações mais complexas como diferenciação e simplificação de expressões.

A interface é um conjunto de programas que manipula vários aspectos da interação do sistema com o usuário. Permite que o usuário especifique um problema que deseja resolver usando uma liguagem; a especificação é interpretada no núcleo que ativa os módulos necessários para a solução do problema e finalmente a solução do problema é encaminhada à interface que se responsabiliza em exibi-la para o usuário.

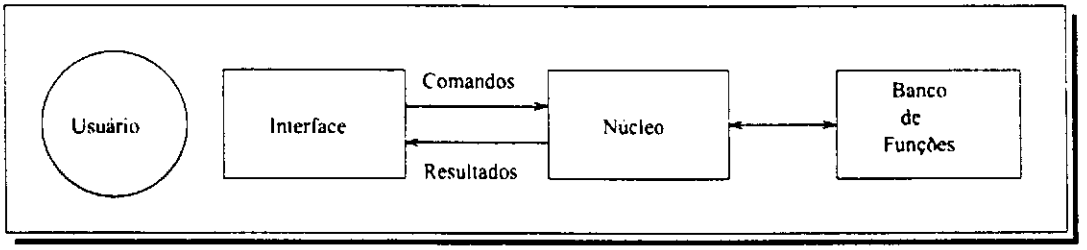

Figura 3.1: Divisão de Sistemas Algébricos

A abordagem estruturada de conceber soluções de problemas conduz naturalmente à modularização do software. Os módulos implementam funções bem definidas e permitem alcançar maior clareza na estrutura lógica do software. A idéia de modularização está associada à idéia de isolar operações ou conjunto de operações e de usar parâmetros em um software. A parametrização de um software pode ser usada para coletar parâmetros de hardware e de software que são usados em diferentes unidades de programa e tornar flexíveis as representações de dados. O isolamento de operações e sua implementação em um módulo permite executar essas operações em diferentes unidades de programa apenas ativando o módulo correspondente. Além disso, é possível conceber o sistema de forma incremental, pela adição de mais recursos, ou seja, novos módulos, podendo até tornar o sistema um especialista numa determinada área. O sistema de software MAPLE foi concebido de forma incremental e hoje conta com mais de 900 funções.

No desenvolvimento de programas é preciso lançar mão de todos os recursos para aumentar a produtividade e a qualidade do software desenvolvido. Para ilustrar a idéia exposta, de modelo de arquitetura de um software algébrico é apresentado na figura 3.3 (não se sabe se é possível construir). Pelo menos conceitualmente é mais fácil visualizar o sistema composto de módulos; para facilitar a compreensão, o modelo foi dividido em três níveis:

- **Nível 1.**

  O módulo **interface do usuário**, deve permitir a utilização de uma variedade eclética de hardware de entrada e saída incluindo o tradicional teclado, resposta audível, voz, *mouse*, digitalizador, pena magnética, vídeo a cores de alta resolução, traçador de gráficos a cores, etc. Também deve dispor de uma linguagem de comunicação com o sistema (comandos do sistema) e uma linguagem de especificação de problemas com o qual o usuário vai dizer ao sistema qual é o seu problema, onde estão os dados, como eles serão fornecidos e como quer a saída.

  As operações que dependem do ambiente devem ser agrupadas em um mesmo módulo (módulo de interface do usuário). As rotinas devem ser organizadas em camadas, uma camada com funções especificamente para cuidar da parte física da máquina. Como é mostra a figura 3.2, as rotinas de aplicação fazem chamadas a um conjunto de funções de interface que devem executar tarefas específicas do equipamento.

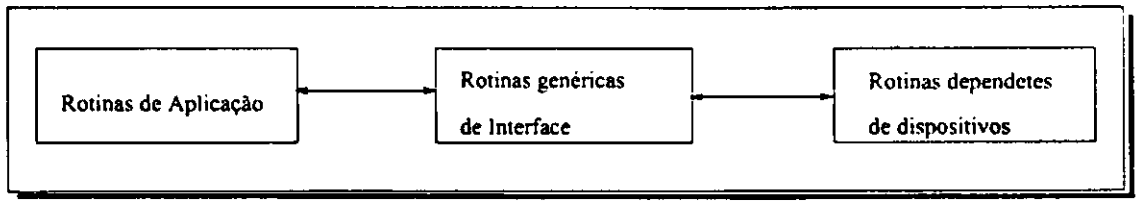

Figura 3.2: Funções dependentes de equipamento abordadas com o conceito de camadas

  Ainda no mesmo nível encontra-se o que pode ser chamado de **interpretador de linguagem**, que deve ter recursos para entender e interpretar corretamente a linguagem utilizada pelo usuário para especificar um problema.

- **Nível 2.**

  Contém o núcleo formado pelo **gerenciador de módulos**, cuja função é identificar o problema do usuário e selecionar métodos baseados nas características

desse problema e assim acionar os módulos adequados. Para isso deve-se embutir alguma inteligência no gerenciador para que ele possa executar corretamente suas atividades.

Devido às limitações do hardware, um sistema algébrico precisa realizar as operações aritméticas por meio de software; o módulo que realiza essas operações (**módulo básico**) e o módulo que armazena os objetos (**módulo de armazenamento**) para realização dessas operações devem também estar no núcleo, comunicando-se entre eles e com o gerenciador de módulos.

Deve conter o **banco de módulos**, formado a partir de pequenos módulos (descritos na seção 3.2), composto de funções que serão ativadas de acordo com as especificações do problema. É comum o usuário programar o seu trabalho usando uma linguagem de alto nível. Esse programa possivelmente deverá ser traduzido para um programa em uma linguagem de nível mais baixo e para isso haverá necessidade de um gerador de programas baseado em gabaritos armazenados em um **banco de gabaritos** e em transformações (precisa de um **banco de transformações**). O usuário poderá querer aproveitar o que já programou anteriormente em linguagem de alto nível ou mesmo executar novamente uma tarefa que já está programada. Para isso, deve ter acesso ao **banco de histórico**.

- **Nível 3.**

  Finalmente, o programa será **compilado**, **ligado** (*link-edited*) e **executado**. Resultados gerados por outros programas poderão servir de dados para um novo programa, o que exige que os resultados obtidos pelo usuário sejam mantidos em um **banco de resultados**.

O capítulo 2 discutiu a tendência atual de integrar recursos de computação gráfica, computação numérica e computação algébrica. Sistemas de software para manipulação algébrica como o *Mathematica* e o MAPLE, tratam a computação gráfica, avaliação de funções e computação algébrica de maneira unificada. Por exemplo, a fim de calcular uma derivada de uma função $f(x)$ para um determinado valor a do seu domínio, primeiro encontra a derivada de $f(x)$ ($f'(x)$), usando manipulação simbólica e a seguir calcula-se o valor numérico $f(a)$. Algumas vezes é desejável obter o seu gráfico.

Para que seja possível promover a integração entre a computação gráfica, a computação numérica e a computação algébrica, esses recursos devem:

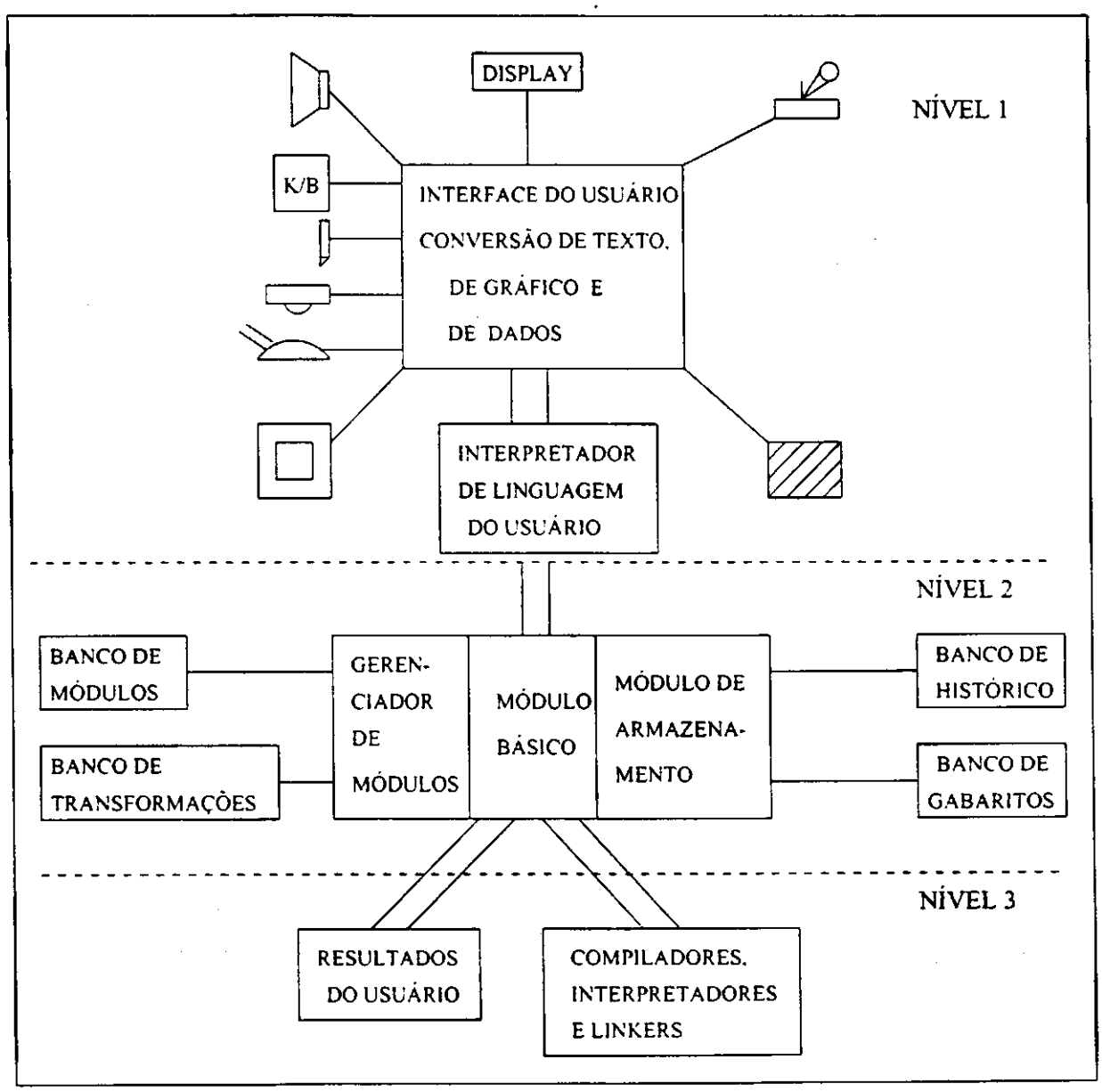

Figura 3.3: Modelo da arquitetura de um sistema de software algébrico

- **Na computação numérica:**
  - o Manusear números de qualquer precisão.
  - o Fazer cálculos matemáticos não apenas com números mas também com objetos como vetores e matrizes.
- **Na computação simbólica:**
  - o Manipular formas algébricas.
  - o Realizar expansão e simplificação de expressões.
  - o Obter soluções analíticas.
- **Na computação gráfica :**
  - o Produzir gráficos de funções, bem como visualização de dados e resultados.

3.2 Módulos de um Sistema

Um software combina diversos algoritmos para que alcance seu objetivo. As funções usadas podem ser agrupadas de acordo com a sua finalidade. Existem algumas operações que são necessárias para todos os demais trabalhos, como as quatro operações com inteiros e polinômios. Assim, haverá o que se pode chamar de **módulo básico** ou **grupo de algoritmos básicos**, aquele que deverá conter as rotinas que executam as operações básicas seja qual for a aplicação.

O grupo básico pode ser assim descrito:

Módulo dos Polinômios e Inteiros ou Módulo Básico

a) Inteiros e Polinômios de uma variável

Os inteiros e polinômios de uma variável podem ser representados como um somatório

$$\sum_{i=0}^{n} a_i b^i.$$

Faz sentido manipulá-los da mesma forma. As operações são:

- Adição/Subtração:

- Multiplicação;
- Divisão (Resultados inteiros, Notação: $a$div$b$);
- Obtenção do resto (Notação: a mod b);
- Potenciação;
- Obtenção do máximo divisor comum (MDC);

b) Polinômios

Para polinômios de grau g em n varáveis deve-se executar operações estruturais, que podem ser consideradas como simplificação de polinômios:

- Expansão;
- Organização, isto é, associação e adição dos termos comuns;
- Fatoração em relação a uma dada variável;

Ainda podem ser executadas operações:

- Adição/Subtração;
- Multiplicação;
- Divisão;
- Operação modular;
- Avaliação polinomial;
- Obtenção das raízes de um polinômio;
- Obtenção do máximo divisor comum (MDC);

É possível organizar do módulo básico da maneira como está disposto na figura 3.4. Os dados são armazenados no módulo de armazenamento, utilizando rotinas para armazenamento de dados, de acordo com a estrutura escolhida para a representação dos objetos. Os objetos são transmitidos ao módulo básico onde são executadas as

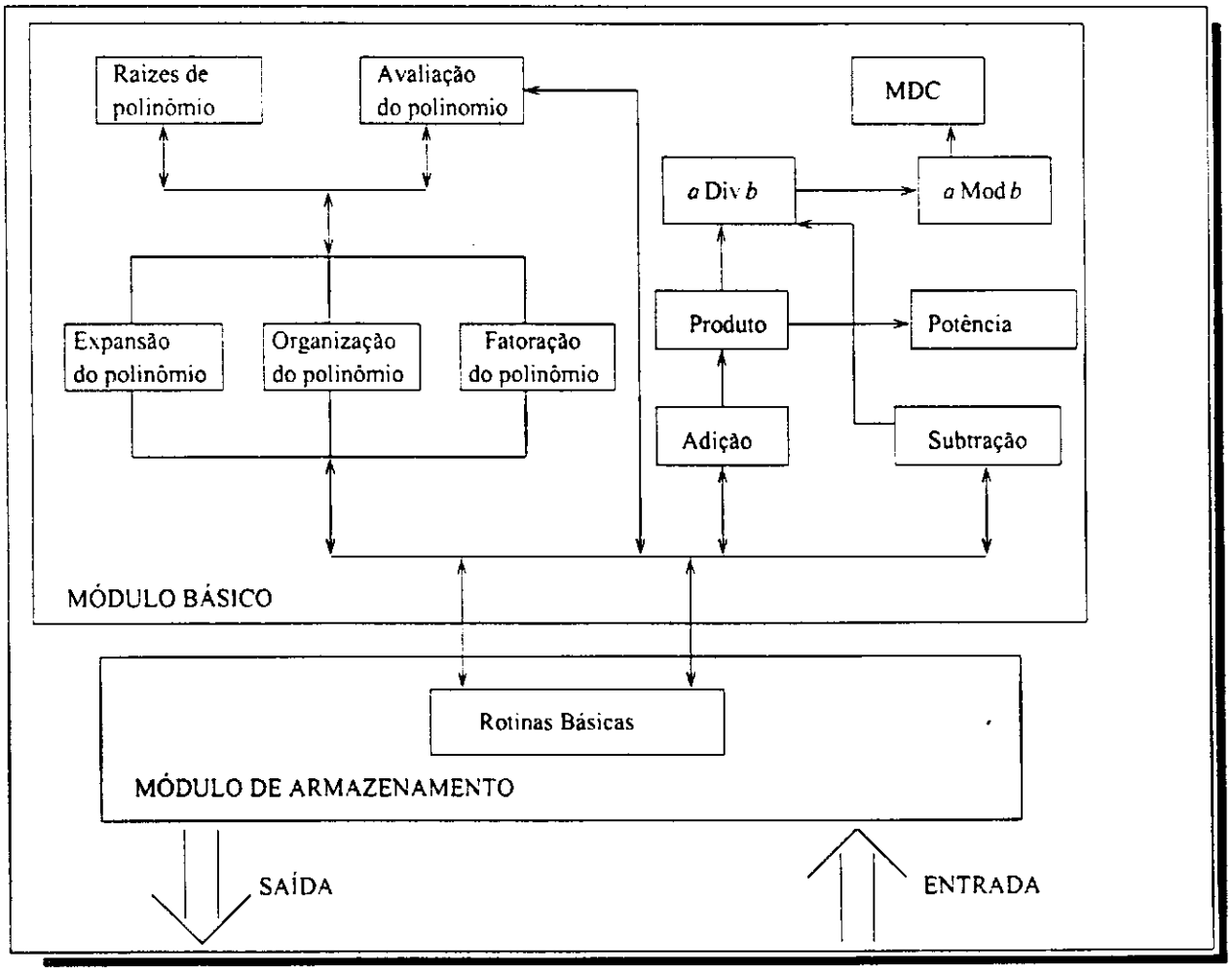

Figura 3.4: Dependência das operações no módulo básico

operações estruturais e aritméticas, ao final da operação o módulo de armazenamento libera os dados para serem manipulados pelo sistema.

Os algoritmos para execução das operações de adição, subtração, multiplicação e divisão serão estudados no capítulo 4.

Outros módulos podem ser adicionados ao banco de módulos. Alguns módulos são descritos para ilustrar a idéia da composição desses módulos.

Módulo para números e polinômios racionais

Os racionais podem ser representados como quociente de dois somatórios.

$$\frac{\sum_{i=n_1}^{m_1} a_i b^i}{\sum_{i=n_2}^{m_2} c_i b^i}$$

Qualquer operação com racionais pode, então, ser realizada com o auxílio do módulo básico.

Para polinômios racionais, independente do número de variáveis e do grau, existem operações que executam a simplificação:

- Expansão e fatoração do denominador;
- Expansão e fatoração do numerador;
- Simplificação do numerador e do denominador.

As operações aritméticas que podem ser executadas são:

- Adição/Subtração;
- Multiplicação;
- Divisão;
- Potenciação;
- Obtenção do recíproco.
- Extração de raiz.

Módulo dos números irracionais

Os números irracionais não podem ser representados com o número finito de dígitos em um computador. A idéia é trabalhar, sempre que possível, com esses números simbolicamente (e, π, $\sqrt{2}$). Caso isso não seja possível será usado o arredondamento (veja nota de rodapé no capítulo 4). O arredondamento deve ser feito de modo que garanta que o erro seja o menor possível.

As operações aritméticas a serem executadas serão as quatro operações básicas.

Como o conjunto dos números reais contém inteiros, racionais e irracionais, o módulo dos números reais é obtido através da integração dos três módulos (inteiros, racionais e irracionais). A seção 4.9 do capítulo 4 discutirá as operações aritméticas básicas com os números reais, introduzindo o conceito de representação em *ponto fixo* para um número *real*.

Módulo de aritmética complexa

Os números complexos podem ser considerados como o produto de um número real por i com $i^2 = -1$, somado a outro número real. As operações aritméticas a serem executadas serão as quatro operações básicas, além da obtenção do conjugado, da partes real e imaginária de um número complexo.

Módulo para diferenciação

O módulo para diferenciação deverá conter as seguintes regras básicas:

- da Constante;
- da Identidade;
- da Potência;
- da Homogenidade;
- da Soma;
- do Produto;
- do Quociente;
- da Potência para expoentes inteiros.

Regras que usam as regras básicas:

- da Cadeia;
- da Função inversa;
- da Raiz;
- da Potência para expoentes racionais.

Deverá diferenciar funções especiais, como:

- Trigonométricas e suas inversas;
- Logarítmicas;
- Exponenciais;
- Hiperbólicas.

Utilizando as regras de diferenciação e a diferenciação de funções especiais, o módulo deve posibilitar o calculo das derivadas parciais e derivadas de ordem superior de polinômios e de funções especiais.

Módulo para obtenção de primitivas

Deverá conter as regras básicas, da soma, da constante, etc, e deve obter a primitiva de funções especiais como

- Trigonométricas e suas inversas;
- Exponencias e logarítmicas;
- Hiperbólicas;
- Funções que envolvem produtos de potências de funções trigonmétricas;
- Funções racionais.

Deve executar as técnicas de integração:

- Por substituição trigonométrica;
- Por partes.

Devem obter a solução de integrais multiplas.

O capítulo 5 apresentará um algoritmo para obtenção de primitivas e fará uma breve comparação de como alguns sistemas algébricos atuais tratam a obtenção de primitivas.

Módulo para problemas envolvendo vetores e matrizes

Deverá realizar operações com vetores (que podem ser considerados como matriz linha ou matriz coluna):

- Adição;
- Mutilplicação por um escalar;
- Produto interno;
- Norma e distância em C^n (Espaço vetorial com n componentes complexos);
- Produto tensorial.

Operações com matrizes:

- Adição;
- Multiplicação por um escalar;
- Multiplicação;
- Transposição;
- Inversão;
- Obtenção do determinante.

Para executar operações como diferenciação e integração de determinadas funções é preciso que o sistema de computação algébrica possa trabalhar com as propriedades e identidades dessas funções; assim é conveniente que o sistema possua módulos que manipulem essas identidades e propriedades.

Módulo para funções trignométricas e suas inversas

- Identidades trigonométricas;
- Funções trigonométricas inversas;
- Identidades envolvendo funções trigonométricas inversas.

Módulo para funções logarítmicas, exponenciais e hiperbólicas

- Propriedades da função exponencial;
- Propriedades das funções logarítmicas;
- Conversão de base para logarítmos;
- Relações entre funções hiperbólicas.

6) Módulos para algumas funções especiais e operações especiais

- Funções de Bessel;
- Transformada de Fourier;
- Transformada de Laplace;
- Funções Zeta, etc.

Como já foi dito, o gerenciador de módulos (ver Figura 3.3) deverá ativar os módulos necessários à solução do problema fornecido pelo usuário. Por exemplo, para diferenciar a função $\tan x \cos x$, além dos módulos que se encontram no núcleo deverão ser ativados os módulos de diferenciação e de funções trignométricas e suas inversas.

No módulo de funções especiais nada impede que as funções presentes sejam agrupadas de uma outra forma. É conveniente também a adição de um módulo de estatística e de um módulo para que possibilite o cálculo de autovalor, transformações lineares, etc.

3.3 Interface

A interface do usuário deve funcionar como um elo entre o programa de aplicação e o usuário, permitindo que esse personalize a exibição das informações de acordo com suas preferências pessoais.

A concepção de interfaces de sistemas com os usuários vem merecendo a atenção de muitos especialistas desde o início da década de 80 [1]. Ainda não foram estabelecidos princípios gerais que possam nortear a construção de interfaces com o usuário, embora o bom senso indique que alguns requisitos devem ser satisfeitos. Um desses requisitos é a interface ser amigável. A dificuldade está no fato de que uma interface amigável para um usuário pode não sê-lo para outro ou um mesmo usuário que considera uma interface amigável quando começa a usar um sistema pode considerá-la não amigável quando se torna experiente na utilização do sitema.

Uma interface ideal é aquela que se adapta às necessidades do usuário. Para um novato deve fornecer muita orientação na utilização de um sistema, mas para um veterano deve apenas evitar uma utilização errônea. Uma interface deve ser inerentemente interativa e possibilitar um diálogo contínuo e dinâmico entre usuário e aplicação. Deve ser controlada por eventos, com as ações do usuário criando ou destruindo as informações exibidas. Quando a aplicação muda de modo, o método que a aplicação usa para comunicar-se com o usuário deve permanecer o mesmo. A consistência de comandos reduz a quantidade de detalhes a ser memorizada para que o usuário opere o sistema, tornando o programa mais fácil e agradável de usar. Um diálogo amistoso constante com o usuário deve ser mantido através de uma combinação de janelas e menus. Finalmente, deve-se usar uma interface e linguagem comum com a qual novas aplicações possam ser facilmente desenvolvidas e testadas.

Como conceber uma interface com tais características é a questão. Embora seja evidente a necessidade de utilizar técnicas de inteligência artificial, não é nada evidente em que conhecimentos deve se basear.

Questões mais concretas no desenvolvimento da interface do usuário são a exploração racional de novos recursos de software, como X-Windows, a utilização de novos recursos de hardware de entrada, como tendência atual para eliminar o tradicional teclado, e a portabilidade do sistema resultante.

3.4 Linguagem

Uma outra questão que se apresenta é a da linguagem que o usuário usará para especificar o seu problema e fornecer os dados necessários. O ideal é uma linguagem que seja a mais próxima daquela usada na área; se a área for restrita essa exigência pode ser satisfeita razoavelmente: se a área for ampla as dificuldades podem crescer muito. Mesmo em uma área restrita como por exemplo a algébra linear numérica já deve apresentar uma diversidade de recursos. De qualquer maneira ao usuário sempre resta o encargo de aprender uma nova maneira de especificar os problemas de sua área.

A questão é facilitar, de maneira econômica, o trabalho do usuário em suas atividades. Para facilitar o trabalho do usuário é necessário uma documentação, clara e objetiva.

- **Manual do usuário**: um ou mais volumes contendo informações variadas, como por exemplo a descrição detalhada dos parâmetros que devem ser fornecidos como dados de um problema e a possível resposta obtida.
- **Um sistema de ajuda on-line**: a documentação é disponível em um terminal ou saída impressa em resposta a comandos emitidos pelo usuário.

Em um sistema algébrico há dificuldade em alcançar a uniformidade de linguagem, principalmente porque os sistemas algébricos são utilizados em diversas áreas e, como já foi dito, a notação matemática depende muito do contexto em que está sendo usada.

Em um software matemático (ver definição 1.1.1) é preciso distinguir a linguagem usada no desenvolvimento do software numérico ou algébrico (ver definição 1.1.2 e 1.1.3) e a linguagem usada pelo usuário. Claro que o usuário precisa dispor de uma linguagem para dizer ao sistema o que deseja fazer, se o requisito dispensa programação precisa ser satisfeito. O ideal é que essa linguagem seja a mais próxima possível da forma como os problemas de matemática são enunciados e solucionados em ciência e tecnologia. Como o método de solução pode ser algébrico, gráfico ou numérico, a linguagem deve fornecer recursos para especificar qual o método a ser usado. Ao criar uma interface há necessidade de inventar uma liguaguem de especificação de problemas de matemática (LAPEM).

Uma LAPEM deve ter recursos para

- criar objetos matemáticos como vetores e matrizes;

- operar com objetos matemáticos criados, como somar vetores ou escalares, multiplicar uma matriz por um vetor ou multiplicar matrizes gerando novos objetos ou não; e
- resolver problemas, como obter o vetor que satisfaz um sistema linear $Ax = b$ em que a matriz A e o vetor b são conhecidos (a solução podendo ser algébrica ou numérica).

A linguagem usada pelo usuário em um software algébrico difere das linguagens convencionais, principalmente com respeito aos *valores* das variáveis (ou vetores ou matrizes) declarados no programa que são expressões algébricas representadas formalmente na memória do computador. Quando um programa contém a instrução para executar a soma de duas variáveis, as expressões simbólicas que são os valores dessas variáveis são adicionadas de acordo com as regras tradicionais da álgebra. O resultado da soma é também uma expressão contendo letras, digitos, parênteses, etc.

Deve existir uma preocupação em como o usuário vai dizer o que fazer e como exibir os resultados, pois qualquer expressão algébrica pode ser representada em mais de uma forma: por exemplo, $3x$ pode ser representado como $x + x + x$. As representações no computador não devem ser ambíguas; geralmente são usadas funções com nomes de operação ou de símbolos usuais da matemática abreviados e com parâmetros bem definidos. Por exemplo, GCD[.... ...] para máximo divisor comum e SUM[..., ...] para somatório.

Alguns sistemas também numeram suas entradas e saídas permitindo que o usuário se refira à expressão pelo número ou use comandos que indiquem a expressão de interesse do usuário. Em alguns sistemas, funções não disponíveis no sistema podem ser implementadas pelo usuário usando uma linguagem de programação embutida no software.

Capítulo 4

Operações Aritméticas Básicas

No capítulo 3 foi apresentada uma organização de um sistema algébrico e foi discutida a sua modularização. A dependência das operações no módulo básico foi apresentada. Neste capítulo serão discutidos algoritmos para a execução das quatro operações aritméticas, incluindo uma estrutura de dados para representar inteiros e polinômios e finalmente os requisitos de uma linguagem para a implementação desses algoritmos. O capítulo pressupõe conhecimentos de transformada de Fourier, de relação de equivalência e de algumas estruturas algébricas. O apêndice trata desses assuntos de forma suscinta e sugere referências bibliográficas.

4.1 Inteiros e polinômios de uma variável.

Em um sistema de software algébrico é inadmissível introduzir erros na computação com números. Quando operações são realizadas em ponto flutuante os resultados não são exatos, já que frequentemente precisam ser arredondados para o ponto flutuante mais próximo [28].

Um sistema de software algébrico deve extrapolar a capacidade de armazenamento do computador, trabalhando com precisão arbitrária. Todas as operações aritméticas precisam ser executadas pelo próprio sistema e não pelo hardware. É necessária a implementação de programas que executem essas operações.

A semelhança entre inteiros positivos e polinômios de uma variável está na representação de ambos como um somatório $\sum_{i=0}^{n-1} a_i x^i$. No caso dos inteiros, os a_i's podem ser escolhidos do conjunto $\{0, \ldots, B-1\}$ e $x = B$, em que B é a base na qual o inteiro

é representado. No caso dos polinômios, os a_i's podem ser escolhidos de um conjunto de coeficientes e x é uma incógnita [29].

Outra semelhança de fundamental importância, é o fato de que o conjunto dos inteiros Z e o conjunto $K[x]$ de polinômios de uma variável sobre o corpo K, serem domínios euclidianos (ver definição no Apêndice). Por isso o estudo desses dois objetos pode ser unificado.

A idéia sobre a representação de inteiros na base B encontra uma aplicação na seguinte forma: O hardware do computador pode executar operações com inteiros cujo tamanho não excede o tamanho da palavra (w) da máquina. Para acomodar inteiros de tamanho maior que w, são necessários recursos de manipulação e uma representação que possibilite o armazenamento do inteiro em uma palavra. Assim torna-se interessante representar o inteiro na forma chamada de *precisão arbitrária*, ou seja, para um inteiro a, escreve-se:

$$a = (a_{n-1} \ldots a_1 a_0)_B$$

em que $B \leq w$.

Dada uma base fixa B, se o número de dígitos de um inteiro $a = a_{n-1} \ldots a_1 a_0$ for n, então a é um inteiro de n dígitos ou tem *comprimento* n. Claramente $a < B^n$. Da mesma forma se $a < B^n$, então a terá no máximo n dígitos.

4.2 Adição e subtração

A bibliografia pesquisada sugere que para executar adição e subtração de dois inteiros a e b, se for necessário, deve-se acrescentar dígitos à esquerda de um deles para que a e b possuam o mesmo comprimento. Assume-se que os números envolvidos neste estudo sejam positivos.

Uma análise dos algoritmos mostra que geralmente é melhor usar rotinas diferentes para realizar as operações de adição e subtração a fim de minimizar o número de operações aritméticas [30].

Para $a = \sum_{i=0}^{n-1} a_i B^i$ e $b = \sum_{i=0}^{n-1} b_i B^i$, em que B é a base, têm-se:

$$a + b = \sum_{i=0}^{n-1} (a_i + b_i) B^i,$$
$$a - b = \sum_{i=0}^{n-1} (a_i - b_i) B^i.$$

Dados a e b pertencentes a um domínio euclidiano (ver definição no Apêndice), com $b \neq 0$, pode-se dividir a por b e obter q e r tais que $a = qb + r$, para $0 \leq r < \nu(b)$, em que $\nu(b)$ é a função do domínio euclidiano . O processo de obtenção de q e r é conhecido como algoritmo de divisão, que se supõe conhecido.

Adição

Para encontrar a soma $a + b$, note que pelo algoritmo de divisão, existem inteiros C_0 e s_0 tal que

$$a_0 + b_0 = C_0 B + s_0, \qquad 0 \leq s_0 < B.$$

C_0 é o *transporte* e só pode assumir o valor 0 ou 1.

Se o algoritmo for aplicado recursivamente, é possível encontrar C_i e s_i para $1 \leq i \leq n-1$ de maneira que

$$a_i + b_i + C_{i-1} = C_i B + s_i, \qquad 0 \leq s_i < B$$

Finalmente, tem-se $s_n = C_{n-1}$, já que a soma de dois inteiros de n dígitos tem no máximo $n+1$ dígitos, obtendo-se então $a + b = (s_n s_{n-1} \ldots s_1 s_0)_B$.

Algoritmo 1: A adição de dois inteiros não negativos $a = (a_{n-1}, \ldots, a_0)_B$, e $b = (b_{n-1}, \ldots, b_0)_B$.

$C := 0$; $i := 0$;

while $i < n$ **do**

$s[i] := \text{resto}(a[i] + b[i] + C, B)$

$C := \lfloor \frac{a[i]+b[i]+C}{B} \rfloor$ [1];

$i := i + 1$

endwhile □

Subtração

[1] Se x é um número real:

- $\lfloor x \rfloor$ = maior inteiro menor ou igual a x
- $\lceil x \rceil$ = menor inteiro maior ou igual a x

Para encontrar a diferença, $a - b$, $a > 0$ e $b > 0$ com $a > b$, usa-se o fato de que no algoritmo de divisão existem inteiros P_0 e d_0 tais que

$$a_0 - b_0 = P_0 B + d_0, \qquad 0 \leq d_0 < B.$$

P_0 é o *empréstimo* e só pode ter valor 0 ou -1.

Se o algoritmo for aplicado recursivamente, é possível encontrar P_i e d_i para $1 \leq i \leq n - 2$ de maneira que

$$a_i - b_i + P_{i-1} = P_i B + d_i, \qquad 0 \leq d_i < B$$

Finalmente, tem-se $P_{n-1} = 0$, desde de que $a > b$, obtendo-se então $a - b = (d_{n-1} \ldots d_1 d_0)_B$.

Algoritmo 2: Subtração de dois inteiros não negativos $a = (a_{n-1}, \ldots, a_0)_B$, e $b = (b_{n-1}, \ldots, b_0)_B$, com $a > b$.

$i := 0$; $P := 0$

while $i < n$ **do**

if $a[i] < b[i]$

$d[i] := \text{resto}(a[i] + B - b[i] + P, B)$

$P := -\left\lceil \frac{a[i]+B-b[i]+P}{B} \right\rceil$;

else

$d[i] := \text{resto}(a[i] - b[i] + P, B)$

$P := \left\lfloor \frac{a[i]-b[i]+P}{B} \right\rfloor$;

endif

$i := i + 1$

endwhile □

É interessante observar que a maioria dos algoritmos discutidos aqui é essencialmente uma mecanização do procedimento manual. Esses algoritmos serão chamados de algoritmos convencionais.

Knuth [30] e Kenneth [31] apresentam algoritmos básicos para inteiros; Lipson [32] estende o estudo também para polinômios de uma variável. Obviamente essas abordagens não têm diferença. A complexidade computacional desses algoritmos, é estabelecida pela seguinte proposição:

Proposição 4.2.1 *[32] : Seja $A(m,n)$ o tempo necessário para somar ou subtrair dois polinômios tendo graus m e n. Então $A(m,n) = O(\max(m,n))$.* □

4.3 Multiplicação

Convencional

Chama-se *deslocamento para esquerda*, a multiplicação de $(a_{n-1}...a_1a_0)_B$ por B^n, onde é necessário apenas deslocar a expressão para esquerda n posições, completando o espaço aberto com n dígitos iguais a zero.

É interessante discutir a multiplicação de um inteiro de n dígitos por um inteiro de um dígito. Para multiplicar $(a_{n-1}... a_1a_0)_B$ por $(b)_B$, usa-se novamente o algoritmo de divisão

$$a_0b = q_0B + p_0, \quad 0 \leq p_0 < B-1 \text{ e } 0 \leq q_0 \leq B-2,$$

já que $0 \leq a_0b \leq (B-1)^2$. Usando novamente a recursividade tem-se

$$a_ib + q_{i-1} = q_iB + p_i, \quad 0 \leq p_i < B-1 \text{ e } 0 \leq q_i \leq B-2.$$

Quando $p_n = q_{n-1}$, obtém-se $(a_{n-1}...a_1a_0)_B\ (b)_B = (p_np_{n-1}... p_1p_0)_B$.

Para multiplicar dois inteiros de n dígitos.

$$\omega = ab = a\left[\sum_{j=1}^{n-1} b_jB^j\right] = \sum_{j=1}^{n-1}(ab_j)B^j.$$

Para cada j multiplica-se a pelo dígito b_j, e então *desloca* para esqueda j dígitos, e finalmente adiciona as n parcelas obtidas, para encontrar o produto.

Knuth propõe um algoritmo em que a operação de *deslocamento* não é necessária, acumulando as parcelas assim que são obtidas. Tem a vantagem de dispensar o armazenamento dos produtos parciais.

Para facilitar o entendimento do algoritmo, os índices dos algarismos de a e b são trocados para $(a_{m-1}... \, a_1 a_0)_B = (a_1... \, a_m)_B$ e $(b_{n-1}... \, b_1 b_0)_B = (b_1... \, b_n)_B$.

Algoritmo 3: Dados dois inteiros não negativos $a = (a_1... \, a_m)_B$ e $b = (b_1... \, b_n)_B$, este algoritmo calcula $ab = (w_1... \, w_{m+n})_B$. Sem perda de generalidade suponha $m \geq n$.

$s := m + 1$;

while $s \leq m + n$ **do** (este loop atribui o valor zero aos termos $w_{m+1} \ldots w_{m+n}$)

$w[s] := 0$;

$s := s + 1$;

endwhile

$i := m$;

while $i > 0$ **do**

if $a[i] = 0$ **then**

$w[i] := 0$;**goto M1**

else $J := n$;

$q := 0$; (Carry)

endif

while $j > 0$ **do**

$t := (a[i] \times b[j]) + w[i + j] + q$

$w[i + j] :=$ resto(t, B);

$q := \lfloor \frac{t}{B} \rfloor$;

$j := j - 1$;

endwhile

$w[i] := q$:

M1: $i := i - 1$;

endwhile □

Proposição 4.3.1 *[32]: Seja $M(m,n)$ o tempo necessário para multiplicar um inteiro de m dígitos por um inteiro de n dígitos. Então $M(m,n) = O(mn)$.* □

O algoritmo 3 não é a mais rápida maneira de obter o produto, apesar da vantagem de sua simplicidade. Por exemplo, quando $m = n = 4$, é possível executar a multiplicação em menos tempo que o tempo requerido pelo algoritmo 3 [30]. Outro método será discutido em seguida.

O Algoritmo da Transformada Rápida de Fourier

A transformada de Fourier é usualmente definida sobre o conjunto dos números complexos. Entretanto, aqui a transformada de Fourier será definida sobre um anel comutativo (ver definição no Apêndice) arbitrário $\langle R, +, \cdot, 0, 1\rangle$.

Definição 4.3.1 *[29]: Um elemento ω de R tal que:*

1. $\omega \neq 0$,
2. $\omega^n = 1$, e
3. $\sum_{j=0}^{n-1} \omega^{jp} = 0$ *para* $0 \leq p < n$

é chamado raiz principal da unidade. Os elementos $\omega^0, \omega^1, \ldots, \omega^{n-1}$ são as raízes n-ésimas da unidade. □

Por exemplo, $e^{\frac{2\pi i}{n}}$, onde $i = \sqrt{-1}$ é a principal raiz n-ésima da unidade no anel dos números complexos.

Definição 4.3.2 *[29]: Seja $a = [a_0, a_1, \ldots, a_{n-1}]^T$ um vetor de n componentes de R. Assuma que o inteiro n tenha inverso multiplicativo em R e que R tenha uma raiz n-ésima da unidade ω. Seja A uma matriz $n \times n$ tal que $A[i,j] = \omega^{ij}$, para $0 \leq i, j < n$. O vetor $F(a) = Aa$, cujo i-ésimo componente é $b_i = \sum_{k=0}^{n-1} a_k \omega^{jk}$ para $0 \leq i < n$, chama-se Transformada Discreta de Fourier de a.* □

A matriz A é não singular, A^{-1} existe e seu ij-ésimo elemento é da forma $((1/n)\omega^{-ij})$.

Definição 4.3.3 *[29]: O vetor $F^{-1}(a) = A^{-1}a$, cujo i-ésimo componente $0 \leq i < n$ é*

$$\frac{1}{n}\sum_{k=0}^{n-1} a_k \omega^{-ik},$$

é a Inversa da Transformada Discreta de Fourier de a. Claramente tem-se $F^{-1}(F(a)) = a$. □

A transformada de Fourier é usada em muitas áreas da ciência e engenharia. É usada na engenharia elétrica na teoria dos códigos, processamento digital de sinais, etc. Em muitas dessas aplicações é conveniente transformar um problema em um outro problema mais fácil. Um exemplo é o cálculo do produto de dois polinômios.

Seja,

$$p(x) = \sum_{i=0}^{n-1} a_i x^i$$

um polinômio de grau $n-1$. Esse polinômio pode ser unicamente determinado de duas formas: pela lista de seus coeficientes $a_0, a_1, \ldots, a_{n-1}$ ou por uma lista de valores em n pontos distintos $p(x_0), p(x_1), \ldots, p(x_{n-1})$, porque a partir desses valores é possível obter os coeficientes do polinômio por uma interpolação polinomial.

Calcular a transformada de Fourier de um vetor $[a_0, a_1, \ldots, a_{n-1}]^T$, é equivalente a converter a representação dos coeficientes de um polinômio $\sum_{i=0}^{n-1} a_i x^i$ para a sua representação pelos valores nas n-ésimas raízes da unidade, $\omega^0, \ldots, \omega^{n-1}$. Da mesma forma a transformada inversa de Fourier, é equivalente a obter um polinômio interpolante conhecidos seus valores nas n raízes da unidade.

Definição 4.3.4 *[29]: Sejam.*

$$a = [a_0, a_1, \ldots, a_{n-1}]^T \text{ e } b = [b_0, b_1, \ldots, b_{n-1}]^T$$

dois vetores coluna. A convolução de a e b, denotada por $a \otimes b$ *é o vetor* $c = [c_0, c_1, \ldots, c_{n-1}]^T$, *onde* $c = \sum_{j=0}^{n-1} a_j b_{j-i}$ *(Tomando* $a_k = b_k = 0$ *se* $k < 0$ *ou* $k \geq n$*).*□

Os $c_i's$ da definição 4.3.4 têm a forma:

$$
\begin{aligned}
c_0 &= a_0 b_0, \\
c_1 &= a_0 b_1 + a_1 b_0, \\
c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0, \\
&\cdots\cdots
\end{aligned}
$$

Note que $c_{2n-1} = 0$; esse termo é incluído apenas por simetria.

Para motivar o estudo da convolução, considere novamente a representação de um polinômio por seus coeficientes. O produto de dois polinômios de grau $n-1$,

$$p(x) = \sum_{i=0}^{n-1} a_i x^i \qquad \text{e} \qquad q(x) = \sum_{j=0}^{n-1} b_j x^j$$

é um polinômio de grau $2n-2$

$$p(x)q(x) = \sum_{i=0}^{2n-2} \left[\sum_{j=0}^{i} a_i b_{i-j} \right] x^i.$$

Os coeficientes do produto são exatamente os componentes da convolução do vetores coeficientes $[a_0, a_1, \ldots, a_{n-1}]^T$ e $[b_0, b_1, \ldots, b_{n-1}]^T$ dos polinômios originais se desprezar c_{2n-1}, que é zero. Isso sugere que a convolução de dois vetores a e b seja a transformada inversa do produto da transformada de dois vetores. Simbolicamente $a \circledcirc b = F^{-1}(F(a) \cdot F(b))$. Existe um algoritmo eficiente para convolução usando a Transformada rápida de Fourier e usando a definição de convolução pode-se executar multiplicação simbólica de polinômios e multiplicação de inteiros de forma mais eficiente.

Avaliar $F(a)$ é equivalente a avaliar o polinômio $p(x) = \sum_{i=0}^{n-1} a_i x^i$ nos pontos $\omega^0, \ldots, \omega^{n-1}$. Como a avaliação de um polinômio $p(x)$ no ponto a é equivalente a encontrar o resto da divisão de $p(x)$ por $x - a$ (escrevendo $p(x) = (x-a)q(x) + c$, onde c é uma constante, então $p(a) = c$). A avaliação da transformada de Fourier se reduz a encontrar os restos da divisão de um polinômio $p(x) = \sum_{i=0}^{n-1} a_i x^i$ de grau $n-1$ por $x - \omega^0, x - \omega^1, \ldots, x - \omega^{n-1}$.

A divisão de $p(x)$ por cada um dos $x - \omega^i$ tem um custo $O(n^2)$. Para obter um algoritmo mais rápido multiplicam-se os $x - \omega^i$'s dois a dois, repetindo o processo com os $n/2$ polinômios resultantes, e assim por diante até obter dois polinômios, q_1 e q_2, cada um dos quais é o produto da metade dos $x - \omega^i$'s. Depois, divide-se $p(x)$ por q_1 e por q_2, obtendo os restos $r_1(x)$ e $r_2(x)$, respectivamente, cada um com grau máximo $(n/2) - 1$. Para cada ω^i para o qual $x - \omega^i$ é fator de q_1, encontrar o resto da divisão de $p(x)$ por $x - \omega^i$ é equivalente a encontrar o resto da divisão de $r_1(x)$ por $x - \omega^i$.

Um procedimento similar deve ser seguido para cada ω^i tal que $x - \omega^i$ seja um fator de q_2. Calcular o resto de $p(x)$ quando dividido por cada um dos $x - \omega^i$'s é equivalente a calcular os restos de $r_1(x)$ e $r_2(x)$ quando dividido por cada um dos $n/2$ $x - \omega^i$'s apropriados. Escolhendo adequadamente a ordem dos $x - \omega^i$, é possível obter os produtos resultantes na forma $x^j - \omega^i$, reduzindo o tempo para multiplicar e dividir polinômios.

Seja $c_0, c_1, ..., c_{n-1}$ uma permutação de $\omega^0, \omega^1, ..., \omega_{n-1}$. Defina os polinômios $q_{l,m}$, para $0 \leq m \leq k$ e l um inteiro multiplo de 2^n no intervalo $0 \leq l \leq 2^k - 1$, da seguinte forma:

$$q_{l,m} = \prod_{j=l}^{l+2^m-1} (x - c_j).$$

$q_{0,k}$ é o produto $(x - c_0)(x - c_1)...(x - c_{n-1})$, e $q_{l,0}$ é, em geral,

$$q_{l,m} = q_{l,m-1} q_{l+2^{m-1},m-1}.$$

A meta é calcular o resto de $p(x)/q_{l,0}(x)$ para cada l. Para fazer isso, calculam-se os restos de $p(x)/q_{l,m}(x)$ para cada $q_{l,m}(x)$ começando em $m = k - 1$ e terminando em $m = 0$.

Suponha que o polinômio resto de $p(x)/q_{l,m}(x)$ de grau $2^m - 1$, $r_{l,m}$, tenha sido calculado. Desde de que $q_{l,m} = q'q''$, onde $q' = q_{l,m-1}$ e $q'' = q_{l+2^{m+1},m-1}$, observa-se que $p(x)/q'(x)$ tem o mesmo resto $r_{l,m}/q'(x)$ e que o mesmo acontece para $q''(x)$.

É possível obter os restos de $p(x)/q'(x)$ e $p(x)/q''(x)$ dividindo o polinômio, $r_{l,m}$, de grau $2^{k-m} - 1$ por $q'(x)$ e $q''(x)$, mais rápido do que dividindo o polinômio, $p(x)$, de grau $2^k - 1$ por $q'(x)$ e $q''(x)$. Escolhendo uma ordenação em que cada c_i é uma potência de ω, é possível garantir que cada polinômio $q_{l,m}$ é da forma $x^{l,m} - \omega^s$ para algum s.

O lema seguinte mostra que escolhendo s =**rev**(j), isto é, se j tiver uma representação binária $[d_0 d_1 ... d_{k-1}]$ então s é o número cuja representação binária é $[d_{k-1} ... d_1 d_0]$, então seja possível obter $q_{l,m}$ da forma $x^{2m} - \omega^s$.

Lema 4.3.1 [29]: *Seja $n = 2^k$ e seja ω a n-ésima raiz principal da unidade. Para $0 \leq j \leq 2^k$, seja $[d_0 d_1 ... d_{k-1}]$ a representação binária do inteiro j e seja* **rev**(j) *o inteiro*

08. end for

09.end for

10.for $l = 0$ **until** $n - 1$ **do** $b_{rev(l)}$:=$r_{l,0}$□

Para calcular a transformada inversa, basta trocar ω por ω^{-1}, e então, dividir $b_{\mathbf{rev(l)}}$ por n.

Exemplo 4.3.1

Dado um vetor a, onde $n = 8 = 2^3$, logo $k = 3$, será encontrado $F(a) = b$ a transformada de Fourier de a. A lista $c_0, c_1, \ldots, c_7$ é igual a $\omega^0, \omega^4, \omega^2, \omega^6, \omega^1, \omega^5, \omega^3, \omega^7$ respectivamente. O $q_{l,m}$ estão ilustrados na figura 4.1.

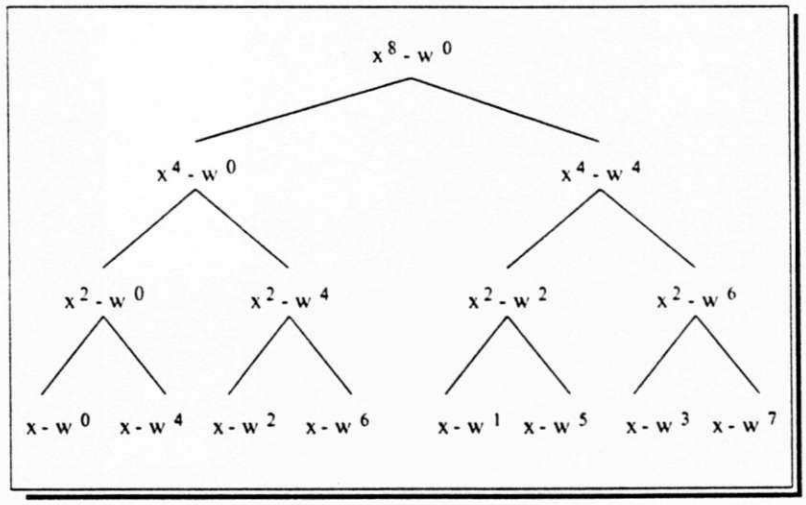

Figura 4.1: Ilustração de $q_{l,m}$ do exemplo 1

Os $q_{l,m}$ são usados para calcular os restos que se seguem.

Inicialmente avaliam-se $r_{0,2}$ e $r_{4,2}$, os restos de $p(x)/(x^4 - \omega^0)$ e $p(x)/(x^4 - \omega^4)$, onde $p(x) = \sum_{j=0}^{7} a_j x^j$.

Para $m = 2$ o laço nas linhas 3-7 é executado apenas para $l = 0$. Na linha 4, $r_{0,3} = \sum_{j=0}^{7} a_j x^j$. Na linha 5, s é 0. E nas linhas 6 e 7,

$$\begin{aligned} r_{0,2} &= (a_3 + a_7)x^3 + (a_2 + a_6)x^2 + (a_1 + a_5)x + (a_0 + a_4) \\ &\epsilon \\ r_{4,2} &= (a_3 + \omega^4 a_7)x^3 + (a_2 + \omega^4 a_6)x^2 + (a_1 + \omega^4 a_5)x + (a_0 + \omega^4 a_4). \end{aligned}$$

Agora avaliam-se $r_{0,1}$ e $r_{2,1}$, os restos da divisão de $r_{0,2}$ por $(x^2 - \omega^0)$ e $(x^2 - \omega^4)$ e $r_{4,1}$ e $r_{6,1}$, os restos da divisão de $r_{4,2}$ por $(x^2 - \omega^2)$ e $(x^2 - \omega^6)$.

Para $m = 1$ o laço nas linhas 3-7 é executado inicialmente para $l = 0$. Na linha 5, s é 0. E nas linhas 6 e 7,

$$\begin{aligned} r_{0,1} &= (a_1 + a_3 + a_5 + a_7)x + (a_0 + a_2 + a_4 + a_6) \\ &\epsilon \\ r_{2,1} &= (a_1 + \omega^4 a_3 + a_5 + \omega^4 a_7)x + (a_0 + \omega^4 a_2 + a_4 + \omega^4 a_6). \end{aligned}$$

Ainda para $m = 1$ o laço nas linhas 3-7 é executado para $l = 4$. Na linha 5, s é 2. E nas linhas 6 e 7,

$$\begin{aligned} r_{4,1} &= (a_1 + \omega^2 a_3 + \omega^4 a_5 + \omega^6 a_7)x + (a_0 + \omega^2 a_2 + \omega^4 a_4 + \omega^6 a_6) \\ &\epsilon \\ r_{2,1} &= (a_1 + \omega^6 a_3 + \omega^4 a_5 + \omega^2 a_7)x + (a_0 + \omega^6 a_2 + \omega^4 a_4 + \omega^2 a_6). \end{aligned}$$

Finalmente avaliam-se $r_{0,0}$, $r_{1,0}$, $r_{2,0}$, ..., $r_{7,0}$, onde $r_{0,0}$, $r_{1,0}$ são os restos de $r_{0,1}$ divido por $(x - \omega^0)$ e $(x - \omega^4)$, $r_{2,0}$ e $r_{3,0}$, os restos de $r_{2,1}$ dividido por $(x - \omega^2)$ e $(x - \omega^6)$, e assim por diante.

Para $m = 0$ o laço nas linhas 3-7 é executado para $l = 0, 2, 4, 6$.

Quando $l = 0$, nas linhas 6 e 7.

$$\begin{aligned} r_{0,0} &= a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_6 + a_7 \\ &\epsilon \\ r_{1,0} &= (a_1 + a_3 + a_5 + a_7)\omega^4 + a_6 + a_2 + a_4 + a_0. \end{aligned}$$

Quando $l = 2$, nas linhas 6 e 7.

$$\begin{aligned} r_{2,0} &= (a_4 + a_0) + (a_1 + a_5)\omega^2 + (a_6 + a_2)\omega^4 + (a_3 + a_7)\omega^6 \\ &\epsilon \\ r_{3,0} &= (a_1 + a_5) + (a_7 + a_3)\omega^4 + (a_4 + a_0)\omega^6 + (a_6 + a_2)\omega. \end{aligned}$$

Quando $l = 4$, nas linhas 6 e 7.

$$\begin{aligned} r_{4,0} &= a_0 + a_{1\omega} + a_2\omega^2 + a_3\omega^3 + a_4\omega^4 + a_5\omega^5 + a_6\omega^6 + a_7\omega^7 \\ &e \\ r_{5,0} &= a_0 + a_1\omega^5 + a_2\omega^2 + a_3\omega^7 + a_4\omega^4 + a_5\omega^9 + a_6\omega^6 + a_7\omega^{11}, \end{aligned}$$

como ω é uma raiz da unidade, $\omega^8 = 1$, e

$$r_{5,0} = a_0 + a_1\omega^5 + a_2\omega^2 + a_3\omega^7 + a_4\omega^4 + a_5\omega + a_6\omega^6 + a_7\omega^3.$$

Quando $l = 6$, nas linhas 6 e 7.

$$\begin{aligned} r_{6,0} &= a_0 + a_1\omega^3 + a_2\omega^6 + a_3\omega^9 + a_4\omega^4 + a_5\omega^7 + a_6\omega^2 + a_7\omega^5, \\ &= a_0 + a_1\omega^3 + a_2\omega^6 + a_3\omega + a_4\omega^4 + a_5\omega^7 + a_6\omega^2 + a_7\omega^5. \\ &e \\ r_{7,0} &= a_0 + a_1\omega^7 + a_2\omega^6 + a_3\omega^{13} + a_4\omega^4 + a_5\omega^{11} + a_6\omega^2 + a_7\omega^9, \\ &= a_0 + a_1\omega^7 + a_2\omega^6 + a_3\omega^5 + a_4\omega^4 + a_5\omega^3 + a_6\omega^2 + a_7\omega. \end{aligned}$$

Na linha 10, para $l = 0$ até 7 têm-se

$b_{rev(0)} = r_{0,0}$ então, $b_0 = r_{0,0}$.

$b_{rev(1)} = r_{1,0}$ então, $b_4 = r_{1,0}$.

$b_{rev(2)} = r_{2,0}$ então, $b_2 = r_{2,0}$.

$b_{rev(3)} = r_{3,0}$ então, $b_6 = r_{3,0}$.

$b_{rev(4)} = r_{4,0}$ então, $b_1 = r_{4,0}$.

$b_{rev(5)} = r_{5,0}$ então, $b_5 = r_{5,0}$.

$b_{rev(6)} = r_{6,0}$ então, $b_3 = r_{6,0}$.

$b_{rev(7)} = r_{7,0}$ então, $b_7 = r_{7,0}$.

Assim, $F(a) = [b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7]^T = [r_{0,0}, r_{4,0}, r_{2,0}, r_{6,0}, r_{1,0}, r_{5,0}, r_{3,0}, r_{7,0}]^T$ □

O problema de calcular o produto de polinômios de uma única variável é realmente o mesmo de calcular a convolução de duas seqüências.

$$\left(\sum_{i=0}^{n-1} a_i x^i\right)\left(\sum_{j=0}^{n-1} b_j x^j\right) = \sum_{k=0}^{2n-2} c_k x^k,$$

onde $c_k = \sum_{i=0}^{n-1} a_m b_{k-m}$.

Como antes, a_p e b_p são iguais a zero se $p < 0$ ou $p \geq n$. Também c_{2n-1} deve ser zero. Observa-se que o coeficiente do produto de dois polinômios de grau n pode ser calculado em $O(n \log n)$ operações.

No próximo exemplo é feito o produto de dois polinômios utilizando método manual e depois aplicando o FFT.

Exemplo 4.3.2

Sejam $p(x) = x^3 + 2x + 1$ e $q(x) = 5x^3 + x^2$ polinômios de grau 3, $n = 4 = 2^2$ como n é potencia de dois (2^2), tem-se $k = 2$. Os vetores dos coeficientes são $a = [1.2.0.1]$ e $b = [0.0.1.5]$

Usando método manual tem-se:

$$p(x) \cdot q(x) = (\sum_{i=0}^{n-1} a_i x^i) \cdot (\sum_{j=0}^{n-1} b_j x^j) = \sum_{j=0}^{2n-2} (\sum_{k=0}^{j} a_k \cdot b_{j-k} x^j)$$

Para $i, j > 3$, considera-se $a_i = 0$ e $b_j = 0$.

O produto para $n = 4$ é,

$$p(x) \cdot q(x) = \sum_{j=0}^{2n-2} (\sum_{k=0}^{j} a_k \cdot b_{j-k} x^j) = \sum_{j=0}^{6} (\sum_{k=0}^{j} a_k \cdot b_{j-k} x^j).$$

Para $j = 0$, $\sum_{k=0}^{0} a_k \cdot b_{j-k} = a_0 b_0 = 1 \cdot 0 = 0$,

$j = 1$, $\sum_{k=0}^{1} a_k \cdot b_{j-k} = a_0 b_1 + a_1 b_0 = 1 \cdot 0 + 2 \cdot 0 = 0$,

$j = 2$, $\sum_{k=0}^{2} a_k \cdot b_{j-k} = a_0 b_2 + a_1 b_1 + a_2 b_1 = 1 \cdot 1 + 2 \cdot 0 + 0 \cdot 0 = 0$,

$j = 3$, $\sum_{k=0}^{3} a_k \cdot b_{j-k} = a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0 = 1 \cdot 5 + 2 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 = 7$.

$j = 4$, $\sum_{k=0}^{4} a_k \cdot b_{j-k} = a_0 b_4 + a_1 b_3 + a_2 b_2 + a_3 b_1 + a_4 b_0 = 1 \cdot 0 + 2 \cdot 5 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 =$ 10.

$j = 5$, $\sum_{k=0}^{5} a_k \cdot b_{j-k} = a_0b_5 + a_1b_4 + a_2b_3 + a_3b_2 + a_4b_1 + a_5b_0 = 1 \cdot 0 + 2 \cdot 0 + 0 \cdot 5 + 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 = 1$,

$j = 6$, $\sum_{k=0}^{6} a_k \cdot b_{j-k} = a_0b_6 + a_1b_5 + a_2b_4 + a_3b_3 + a_4b_2 + a_5b_1 + a_6b_0 = 1 \cdot 0 + 2 \cdot 0 + 0 \cdot 0 + 1 \cdot 5 + 0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 = 5$.

Portanto,

$$\begin{aligned} p(x) \cdot q(x) &= \sum_{j=0}^{6}(\sum_{k=0}^{j} a_k \cdot b_{j-k}x^j) \\ &= 0x^0 + 0x^1 + 1x^2 + 7x^3 + 10x^4 + 1x^5 + 5x^6 \\ &= 0 + x + x^2 + 7x^3 + 10x^4 + x^5 + 5x^6 \\ &= 5x^6 + x^5 + 10x^4 + 7x^3 + x^2 + x. \end{aligned}$$

Os vetores dos coeficientes são $p = [1, 2, 0, 1, 0, 0, 0, 0]$ e $q = [0, 0, 1, 5, 0, 0, 0, 0]$, observe que para $i, j > 3$, completam-se a_i e b_j com zeros. O produto de dois polinômios usando o FFT, obtido através da convolução $p \circledcirc q = F^{-1}(F(p).F(q))$, onde $F(p), F(q)$ são as transformadas discretas de Fourier dos vetores coeficientes de $p(x)$ e $q(x)$, respectivamente. Usando o algoritmo FFT, obtêm-se:

$$\begin{aligned} F(p) &= [4, 1 + 3\omega^4, 1 + 2\omega^2 + \omega^6, 2 + \omega^4 + \omega 6, 1 + 2\omega + \omega^3, 1 + 2\omega^5 + \omega^7, \\ &\quad 1 + \omega + 2\omega^3, 1 + \omega^5 + 2\omega 7]^T \end{aligned}$$

e

$$F(q) = [6, 1 + 5\omega^4, \omega^4 + 5\omega^6, \omega^2 + 5\omega^4, \omega^2 + 5\omega^3, \omega^2 + 5\omega^7, \omega + \omega^6, 5\omega^5 + \omega^6]^T.$$

Usando o teorema da convolução calcula-se:

$$\begin{aligned} F(p) \cdot F(q) &= [24, 16 + 7\omega^4, 10 + \omega^2 + 6\omega^4 + 7\omega^6, 6 + 7\omega^2 + 10\omega^4 + \omega^6, \\ &\quad \omega^2 + 7\omega^3 + \omega^5 + 10\omega^4 + 5\omega^6, 10\omega + \omega^2 + 10\omega^4 + 5\omega 6 + 7\omega 7, \\ &\quad 3\omega + \omega^2 + 2\omega^4 + \omega^6 + \omega 7, \omega^2 + \omega^3 + 10\omega^4 + 7\omega 5 + \omega^6]^T. \end{aligned}$$

Calcula-se a transformada inversa de Fourier, $F^{-1}(F(a) \cdot F(b))$, fazendo uso do algoritmo FFT, substituindo no algoritmo ω por ω^{-1} e na linha 10 divide-se $b_{\mathbf{rev(l)}}$ por n.

Obtém-se assim o vetor;

$$F^{-1}(F(p) \cdot F(q)) = [0, 1, 1, 7, 10, 1, 5, 0]^T$$

que é exatamente o vetor dos coeficientes do produto de $p(x)$ por $q(x)$.□

Mais detalhes sobre o FFT, podem ser encontrados em Horowitz e Sahni [33] e Baase [34].

4.4 Divisão

Sejam a e b inteiros na base B tendo m e n dígitos, respectivamente . Dividindo-se a por b, no algoritmo de divisão obtém-se q e r tais que:

$$a = bq + r \qquad 0 \leq r < b. \tag{4.1}$$

Se $a < b$ então nenhum cálculo é necessário: $q = 0$ e $r = a$ satisfaz o algoritmo de divisão. Então considere $a \geq b$.

Observando a figura 4.2, no primeiro passo da divisão polinomial (algoritmo convencional) obtém-se $q_{m-n} = b_n^{-1} a_m$ para calcular o c_i 's. Sendo necessárias $n + 1$ multiplicações e n subtrações para calcular o d_i 's. Esse passo é repetido para cada um dos $m - n - 1$ dígitos de $q(x)$, resultando um total de $(n+1)(m-n+1)$ multiplicações, $n(m - n + 1)$ subtrações e uma adição para calcular b_n^{-1}. Dessa análise resulta:

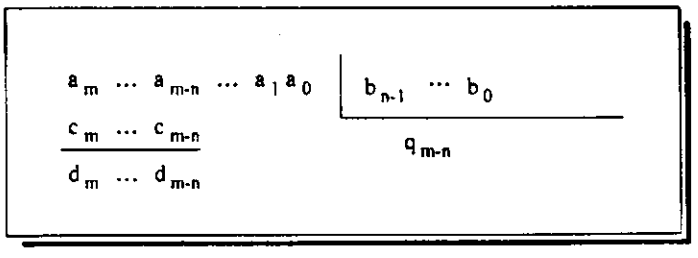

Figura 4.2: Cálculo do quociente para divisão de inteiros

Proposição 4.4.1 *[32]: Seja $D(m, n)$ o tempo necessário para dividir um polinômio de grau m por um polinômio de grau n $(m \geq n)$ a fim de encontrar o quociente de grau $m - n$ e um resto de grau $< n$. Então $D(m, n) = O(n(m - n))$.*□

Para encontrar os q_i's do quociente $q = (q_{n-1}q_{n-2}...q_1q_0)_B$ na divisão polinomial, foram selecionados dois métodos.

Kenneth [31] sugere o seguinte método:

Deseja-se encontrar o q da equação 4.1. Se q for $(q_{n-1}q_{n-2}...q_1q_0)_B$, então

$$a = b\left[\sum_{j=0}^{n-2} q_j B^j\right] + r. \qquad 0 \leq r < b. \tag{4.2}$$

O lado direito da equação 4.2 não é apenas positivo, mas também é menor do que bB^{n-1}. Assim

$$0 \leq a - bq_{n-1}B^{n-1} < bB^{n-1}.$$

permite concluir que,

$$q_{n-1} = \left\lfloor \frac{a}{bB^{n-1}} \right\rfloor.$$

onde $\lfloor x \rfloor$ é a função maior inteiro menor ou igual a x.

É possível obter q_{n-1} subtraindo sucessivamente bB^{n-1} de a até que o resultado obtido seja negativo. Então q_{n-1} será o número de subtrações menos um.

Para encontrar os outros dígitos de q, define-se uma seqüência de *restos parciais* r_i por

$$r_0 = a \qquad \text{e} \qquad r_i = r_{i-1} - bq_{n-1}B^{n-i}$$

para $i = 1, 2, ..., n$, com $0 \leq r_i < b$. Cada dígito q_{n-i} será dado por

$$\left\lfloor \frac{r_{i-1}}{bB^{n-i}} \right\rfloor.$$

Esse algoritmo executa a divisão de um inteiro de $2n$ dígitos por um inteiro de n dígitos em $O(n^2)$ operações. Na verdade o número de operações necessário para uma divisão inteira se aproxima do número de operações necessário para a multiplicação inteira realizada durante o processo de divisão. O seguinte teorema fornece a complexidade computacional da divisão, de acordo com o número de multiplicações.

Teorema 4.4.1 *[31]: Existe um algoritmo para encontrar o quociente $q = \lfloor a/b \rfloor$, de um inteiro a de $2n$ dígitos por um inteiro b não tendo mais que n dígitos, usando $O(M(n))$ operações, onde $M(n)$ é o número de operações necessárias para multiplicar dois inteiros de n dígitos.* □

Um segundo método de obtenção dos q_i's é estudado por Knuth[30].

Para dividir um inteiro de $m+n$ dígitos por um inteiro de n dígitos, o método manual de divisão requer uma certa heurística. Observe que no processo de divisão, o problema é desmembrado em problemas mais simples, onde em cada passo é executada a divisão de um inteiro u de $n+1$ dígitos por um inteiro v de n dígitos, onde $0 < u/v < B$. O resto, menor do que v, é usado no passo seguinte.

Uma estimativa de q baseada nos dígitos mais significativos de u e v, é $\hat{q} = \min(\lfloor \frac{u_0 b + u_1 v_1}{v_1} \rfloor, B-1)$.

Teorema 4.4.2 *[30]: Se $\hat{q} = \min(\lfloor \frac{u_0 b + u_1 v_1}{v_1} \rfloor, B-1)$ então $\hat{q} \geq q$.* □

Teorema 4.4.3 *[30]: Se $v_1 \geq \lfloor \frac{B}{2} \rfloor$, então $\hat{q} - 2 \leq q \leq \hat{q}$.*□

A importância do teorema 4.4.3 é garantir que a estimativa $\hat{q}$ seja independente de B; não importa o valor de B, a estimativa do quociente $\hat{q}$ nunca vai ter um erro maior que duas unidades. O algoritmo 4 usa a escolha de $\hat{q}$ garantindo que $q = \hat{q}$ ou $\hat{q} - 1$.

Algoritmo 4: Dados dois inteiros não negativos $(u_1 u_2 ... u_{n+m})_B$ e $(v_1 v_2 ... v_n)_B$, com $v_1 \neq 0$ e $n > 1$, obtém o quociente $\lfloor u/v \rfloor = (q_0 q_1 q_2 ... q_m)_B$ e o resto $r = (r_1 r_2 ... r_n)_B$. Se $n = 1$, um algoritmo simples de divisão pode ser usado.

$d := \left\lfloor \frac{B}{v_1+1} \right\rfloor$ (Normalização)

if $d = 1$ **then** $u_0 = 0$;

else

$u_0 u_1 ... u_{n+m} := u_1 u_2 ... u_{n+m} \times d$;

$v_1 v_2 ... v_n := v_1 v_2 ... v_n \times d$;

endif

$j := 0$;

while $j \leq m$ **do**

if $u_j = v_1$ **then** $\hat{q} := B - 1$ **else** $\hat{q} := \lfloor \frac{u_j B + u_{j+1}}{v_1} \rfloor$ **endif**

while $v_2\ \hat{q} > (u_j B + u_{j+1} - \hat{q}v_1)B + u_{j+2}$ **do** $\hat{q} := \hat{q} - 1$;**endwhile**

$u_j u_{j+1} ... u_{j+n} := u_j u_{j+1} ... u_{j+n} - (\hat{q} \times v_1 v_2 ... v_n)$

$q_j := \hat{q}$

if $u_j u_{j+1} ... u_{j+n} < 0$ **then**

$q_j := q_j - 1$: $u_j u_{j+1} ... u_{j+n} := u_j u_{j+1} ... u_{j+n} + 0 v_1 v_2 ... v_n$;

else $j := j + 1$ **endif**

endwhile

$r = (r_1 r_2 ... r_n)_B$ é obtido dividindo $u_j u_{j+1} ... u_{j+n}$ por d □

O tempo de processamento pode ser estimado considerando as seguintes variáveis:

M onde $M + 1$ é o número de dígitos no quociente;

N é o número de dígitos no divisor;

E o tempo necessário para q ser ajustado no laço do segundo while;

K e K' os tempos gastos nos ajustes nos *transportes* feitos durante os laços de subtração e multiplicação.

Se assumir que $K + K'$ seja aproximadamente igual a $(N + 1)(M + 1)$, e E aproximadamente igual a $\frac{1}{2}M$, o tempo total gasto no processamento, será em torno de $30MN + 30N + 91M + 113$ vezes $67N + 235M + 4$ se $d > 1$.

4.5 A Escolha da Base

Como já foi dito antes, a representação posicional de inteiros está intimamente relacionada com polinômios. A notação polinomial (posicional) pode ser generalizada para qualquer base $B \geq 2$. O número de m dígitos na base B é $(a_{m-1} ... a_1 a_0)_B$, onde cada a_i escrito na base B é tal que $0 \leq a_i < B$.

Os algoritmos apresentados acima são destinados a uma base arbitrária $B > 1$.

Adição e Subtração

Sejam a e b inteiros positivos na base B de comprimentos m e n, respectivamente. Sem perda de generalidade pode-se assumir que $m \geq n$. Seja $c = a + b$. Como a e $b < B^m$, segue-se que $c < 2B^m$. O comprimento de c será $\leq m + 1$, onde m é o comprimento máximo das parcelas e o 1 deve-se ao possível transporte.

Para executar operações de adição e subtração basta escolher $B = w - 1$, onde w é o maior inteiro representável em complemento de 2, numa palavra; seu valor é $w = 011...1_2 = 2^{k-1} - 1$, onde k é o número de bits de uma palavra.

Multiplicação

Sejam a e b inteiros positivos na base B de comprimentos m e n, respectivamente. Sem perda de generalidade pode-se assumir que $m \geq n$. Seja $c = ab$. Como $a < B^n$ e $b < B^m$, segue-se que $c < B^{m+n}$. O comprimento de c será $\leq m + n$.

Para executar operações de multiplicação deve-se escolher uma base tal que

$$(B-1)^2 \leq 2^{k-1} - 1,$$
$$B \leq \sqrt{2^{k-1} - 1} \cong 2^{\frac{k}{2}-1}.$$

Haverá uma perda de espaço. Entretanto, essa perda será compensada pelo fato do resultado da multiplicação de dois dígitos ser representável em uma palavra, o que implica na possibilidade de usar a operação de multiplicação disponível em hardware.

Divisão

Sejam a e b inteiros positivos na base B de comprimentos m e n, respectivamente. Dividindo-se a por b obtêm-se q e r satisfazendo o algoritmo de divisão. Se $a < b$, nenhum cálculo é necessário. Sem perda de generalidade pode-se assumir que $m \geq n$ e assim $a \geq b$.

Para limitar o comprimento do quociente q, sabe-se que $B^{m-1} \leq a < B^m$ e $B^{n-1} \leq b < B^n$, segue-se que

$$q \leq a/b < B^m/B^{n-1} = B^{m-n+1}.$$

Logo, o comprimento de q será $\leq m - n + 1$.

A divisão é executada por meio de multiplicações e subtrações. Para o produto de dois inteiros a base escolhida deve ser $B \leq 2^{\frac{k}{2}-1}$, que é adequada também para a operação de divisão, já que o comprimento de q não excede o comprimento de c.

Assim, a base escolhida para representar um inteiro e realizar as quatro operações básicas será $B \leq 2^{\frac{k}{2}-1}$.

4.6 Aritmética Modular

Existem certas aplicações nas quais é conveniente realizar a aritimética com inteiros na notação modular (ver definição no Apêndice). Ao invés de representar um inteiro pelos dígitos numa base, representa-se pelos restos da divisão desse número pelos elementos de um conjunto de números primos entre si. Se $p_0, p_1, \ldots, p_{k-1}$ forem inteiros primos entre si e $p = \prod_{i=0}^{k-1} p_i$, então pode-se representar qualquer inteiro u, $0 \leq u < p$, unicamente por um conjunto de restos $u_0, u_1, \ldots, u_{k-1}$ onde $u_i = u$ módulo p_i, para $0 \leq i \leq k$. Quando $p_0, p_1, \ldots, p_{k-1}$ forem conhecidos escreve-se $u \longleftrightarrow (u_0, u_1, \ldots, u_{k-1})$.

É fácil executar a adição, subtração e multiplicação de tal forma que os resultados continuem entre 0 e $p-1$. Sejam,

$$u \longleftrightarrow (u_0, u_1, \ldots, u_{k-1}) \text{ e } v \longleftrightarrow (v_0, v_1, \ldots, v_{k-1}).$$

Então,

$u + v \longleftrightarrow (w_0, w_1, \ldots, w_{k-1})$, onde $w_i = (u_i + v_i) \bmod p_i$;

$u - v \longleftrightarrow (x_0, x_1, \ldots, x_{k-1})$, onde $x_i = (u_i - v_i) \bmod p_i$;

$uv \longleftrightarrow (y_0, y_1, \ldots, y_{k-1})$, onde $y_i = (u_i v_i) \bmod p_i$.

Entretanto, não é claro como executar a divisão usando a aritmética modular. Nem sempre u/v é necessariamente um inteiro, e se for, não é sempre possível encontrar sua representação modular calculando (u_i/v_i) módulo p_i para cada i. De fato, se p_i não for um primo, pode haver muitos inteiros w entre 0 e $p_i - 1$ tais que u_i/v_i módulo p_i satisfaça $wv_i \equiv u_i$ módulo p_i. Por exemplo, se $p_i = 6$, $v_i = 3$ e $u_i = 3$, então ω pode ser 1,2,3, já que $1 \times 3 \equiv 3 \times 3 \equiv 5 \times 3 \equiv 3 \bmod 6$. Assim $(u_i/v_i) \bmod p_i$ não tem sentido.

Uma vantagem da aritmética modular é poder implementá-la utilizando menos operações de hardware que a necessária para a aritmética convencional. Não há necessidade de *transporte*. Infelizmente problemas para executar a divisão e detectar

overflow aparecem constantemente, razão porque as operações com aritmética modular são raramente implementadas.

Para usar a aritmética modular são necessários algoritmos para converter da notação posicional para modular e virce-versa.

Considere o problema de converter um inteiro na notação modular para notação posicional. Sejam $p_0, p_1, \ldots, p_{n-1}$ os módulos relativamente primos e sejam $u_0, u_1, \ldots, u_{n-1}$ os resíduos, em que $n = 2^t$ (t inteiro); deseja-se encontrar o inteiro u tal que $u \longleftrightarrow (u_0, u_1, \ldots, u_{n-1})$. É possível executar essa conversão com os inteiros por uma fórmula análoga à *fórmula de interpolação de Lagrange* para polinômios, de acordo com o seguinte lema.

Lema 4.6.1 [29]: *Seja c_i o produto de todos os p_j's, exceto p_i ($c_i = p/p_i$ onde $p = \prod_{j=0}^{k-1} p_j$). Seja $d_i = c_i^{-1}$ módulo p_i, isto é, $d_i c_i \equiv 1 \bmod p_i$, e $0 \leq d_i < p_i$. Então*

$$u \equiv \sum_{i=0}^{n-1} c_i d_i u_i \bmod p. \qquad \square \tag{4.3}$$

Observando que $u \equiv \sum_{i=0}^{k-1} c_i d_i u_i \bmod p$, nota-se que os termos $c_i d_i u_i$ têm muitos divisores em comum para $0 \leq i \leq n-1$. Por exemplo, $c_i d_i u_i$ tem $p_0, p_1, \ldots, p_{(k/2)-1}$ como um divisor para $i \geq k/2$, e tem $p_{k/2}, p_{(k/2)+1}, \ldots, p_{k-1}$ como um divisor se $i < k/2$. Assim é possível escrever a equação 4.3 como

$$u = (\sum_{i=0}^{(k/2)-1} c_i' d_i u_i) \times \prod_{i=(k/2)+1}^{k-1} p_i + (\sum_{i=(k/2)+1}^{k-1} c_i'' d_i u_i) \times \prod_{i=0}^{(k/2)-1} p_i$$

onde c_i' é o produto $p_0 p_1 \ldots p_{(k/2)-1}$ excluindo p_i, e c_i'' é o produto $p_{k/2} p_{k/2+1} \ldots p_{k-1}$ excluindo p_i, sugerindo que seja o calculado o produto:

$$q_{i,j} = \prod_{m=i}^{i+2^j-1} p_m$$

e, em seguida, sejam calculados os inteiros

$$s_{i,j} = \sum_{m=i}^{i+2^j-1} q_{ij} d_m u_m / p_m.$$

Se $j = 0$, então $s_{10} = d_i u_i$. Se $j > 0$, calcula-se $s_{i,j}$ pela fórmula.

$$s_{i,j} = s_{i,j-1}q_{i+2^{j-1},j-1} + s_{i+2^{j-1},j-1}q_{i+2^{j-1},j-1}.$$

Finalmente calcula-se $s_{0,t} = u$, onde 2^t é o número de termos.

O exemplo 4.6.1, ilustra bem a conversão da aritmética modular para aritmética inteira.

Exemplo 4.6.1

Sejam p_0, p_1, p_2, p_3 iguais a 2, 3, 5, 7, e seja (u_0, u_1, u_2, u_3) igual a (1, 2, 4, 3). $n = 4 = 2^2$, logo $t = 2$. Então $q_{i,0} = p_i$ para $0 \leq i < 4$, $q_{0,1} = 6$, $q_{2,1} = 35$, e $q_{0,2} = p = 210$.

Note que,

$d_0 = (3 \times 5 \times 7)^{-1} \bmod 2 = 1$, já que $1 \times 105 \equiv 1 \bmod 2$,

$d_1 = (2 \times 5 \times 7)^{-1} \bmod 3 = 1$, já que $1 \times 70 \equiv 1 \bmod 3$,

$d_2 = (2 \times 3 \times 7)^{-1} \bmod 5 = 3$, já que $3 \times 42 \equiv 1 \bmod 5$,

$d_3 = (2 \times 3 \times 5)^{-1} \bmod 7 = 4$, já que $4 \times 30 \equiv 1 \bmod 7$.

Então calcula-se,

$$s_{0,0} = 1 \times 1 = 1,\ s_{10} = 1 \times 2 = 2,$$
$$s_{2,0} = 3 \times 4 = 12,\ s_{10} = 4 \times 3 = 12.$$

Para $j = 1$ e $i = 0$ e 2, calculam-se,

$$s_{0,1} = s_{0,0} \times q_{1,0} + s_{1,0} \times q_{0,0} = 1 \times 3 + 2 \times 2 = 7,$$
$$s_{2,1} = s_{2,0} \times q_{3,0} + s_{3,0} \times q_{2,0} = 12 \times 7 + 12 \times 5 = 144.$$

Para $j = 2$, i assume apenas o valor 0. Obtém-se,

$$s_{0,2} = s_{0,1} \times q_{2,1} + s_{2,1} \times q_{0,1} = 7 \times 35 + 144 \times 6 = 1109 = u.\square$$

4.7 Polinômios de Várias Variáveis

Adição e Multiplicação com Polinômio de Várias Variáveis

Um polinômio de r variáveis $a(x_1,\ldots,x_r)$ sobre um corpo F pode ser visto como um polinômio $a(x_1,\ldots,x_r) = \sum_i a_i x_r^i$ de uma variável x_r cujos coeficientes a_i são polinômios em $x_1,\ldots,x_{r-1}$, isto é, $a_i = a(x_1,\ldots,x_{r-1})$. Em resumo é possível enxergar um polinômio de r variáveis como membro de $F[x_1,\ldots,x_{r-1}][x_r]$.

As fórmulas para adição e multiplicação,

$$c(x) = a(x) + b(x) \Leftrightarrow c_k = a_k + b_k \text{ e}$$
$$c(x) = a(x)b(x) \Leftrightarrow c_k = \sum_{i+j=k} a_i b_j,$$

podem ser consideradas extensões recursivas dos algoritmos clássicos para polinômios de uma variável. Se os coeficientes a_k, b_k acima forem polinômios em $r > 0$ variáveis, então a soma e o produto dos coeficientes podem ser obtidos por recursão; se $r = 0$ então essas operações com coeficientes se reduzem a operações no corpo F.

Sejam $a(x_1,\ldots,x_r)$ e $b(x_1,\ldots,x_r)$ polinômios de r variáveis e graus n e m, respectivamente, cada um com r variáveis. Percebe-se que um polinômio $a(x_1,\ldots,x_r)$ de grau m com r variáveis tem $(m+1)^r$ termos e pode ser expandido na forma:

$$a(x_1,\ldots,x_r) = \sum_{e \in N^r} a_i x_1^{e_1} x_2^{e_2} \ldots x_r^{e_r},$$

onde N^r é a r-úpla $(e_1, e_2, \ldots, e_r)$.

Adição

O tempo para calcular $a(x_1,\ldots,x_r) \pm b(x_1,\ldots,x_r)$ irá crescer não mais que o número máximo de termos de $a(x_1,\ldots,x_r)$ ou $b(x_1,\ldots,x_r)$.

Proposição 4.7.1 *[32]: Seja $A(m,n,r)$ o tempo para adicionar (subtrair) dois polinômios com graus m e n e cada um com r variáveis. Então $A(m,n,r) = O(\max(m,n)^r)$.*□

Multiplicação

O tempo para calcular $a(x_1,\ldots,x_r)\ b(x_1,\ldots,x_r)$ vai ser essencialmente o tempo para multiplicar cada um dos $(m+1)^r$ termos de $a(x_1,\ldots,x_r)$ por cada um dos $(n+1)^r$ termos de $b(x_1,\ldots,x_r)$.

Proposição 4.7.2 *[32]: Seja $M(m,n,r)$ o tempo necessário para multiplicar dois polinômios de graus m e n e cada um com r variáveis. Então $M(m,n,r) = O((mn)^r)$.*□

Representação e Manipulação de Polinômios de Várias Variáveis

Collins [35] discute a representação de polinômios de r variáveis, considerando-os como elementos de $Z[x_1,...,x_r]$ onde Z é o conjunto dos inteiros. Como $Z[x_1,...,x_r]$ é isomorfo (ver definição no Apêndice) a $Z[x_1],...,[x_r]$ é possível considerá-lo como polinômio de uma *variável principal*, x_r, com coeficientes que são polinômios nas variáveis x_i com $1 \leq i \leq r-1$. Essa idéia sugere duas diferentes formas de representar e operar com polinômios.

Na primeira forma, escreve-se o polinômio $P(x_1,...,x_r)$ na forma $\sum_{e \in N^r} a_i x_1^{e_1} x_2^{e_2} ... x_r^{e_r}$ onde a_i é um inteiro não nulo e representa P por uma lista $(\alpha_1,...,\alpha_r)$ em que cada α_i é uma lista que representa o i-ésimo termo $a_i x_1^{e_{i1}} x_2^{e_{i2}} ... x_r^{e_{ir}}$. Essa representação é usada no ALPAK [36, 37, 38]. Na segunda forma, escreve-se $P(x_1,...,x_r) = \sum_{i=1}^{m} P(x_1,...,x_{r-1}) x_r^{e_i}$ onde P_i é um polinômio não nulo em $x_1,...,x_{r-1}$, e então usa indução sobre r. P é representado pela lista $((\alpha_1,\beta_1),...,(\alpha_m,\beta_m))$,em que α_i é uma lista representando P_i, e β_i é uma representação na variável x_r e seu expoente e_i. Esse é o método usado no Programa para executar operações com polinômios de várias variáveis com coeficientes inteiros no sistema PM . [35].

A adição, a subtração, a multiplicação e a divisão de polinômios em PM são operações recursivas. O coeficientes inteiros são representados em precisão arbitrária. Cada polinômio é representado de maneira única na forma canônica (ver definição Capítulo 2) escolhida de maneira eficiente pelo programa. Para executar as operações, o sistema manipula os polinômios de maneira que os dois polinômios a serem operados tenham a mesma estrutura, isto é, contenham as mesmas variáveis, na mesma ordem. Collins [35, 39] apresenta mais detalhes sobre as operações com polinômios.

4.8 Estrutura de dados

Na implementação precisam ser resolvidos problemas como o de representar e armazenar objetos. Considerando as semelhanças entre polinômios de uma variável e inteiros, bem como o interesse em manipular os inteiros em precisão arbitrária, uma estrutura de dados para representar esses objetos é sugerida.

Inteiros e Polinômios de uma Variável

Lista encadeada, com os seguintes elementos, estruturados como mostra a figura 4.3:

- Cabeça de lista, contendo o sinal do inteiro, o tamanho da lista, apontador para o início da lista e apontador para o final da lista.
- Nó de lista, contendo o apontador para o nó de dado e o apontador para o próximo nó.
- Nó de dado, contendo um coeficiente do termo do somatório e o expoente do termo do somatório correspondente a esse coeficiente.

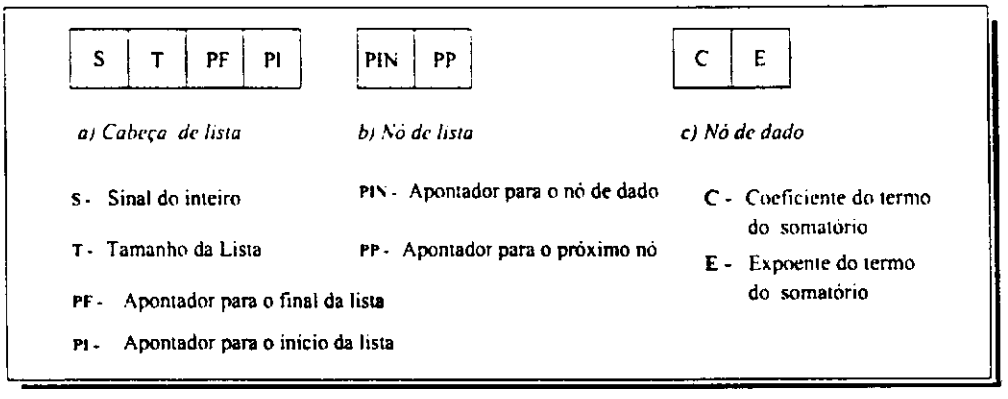

Figura 4.3: Elementos de uma lista para o armazenamento de inteiros

O inteiro $x = \sum_{i=0}^{n} a_i B^i$ escrito como um somatório ou o polinômio $p(x) = \sum_{i=0}^{n} a_i x^i$ podem ser representados na lista como mostra a figura 4.4.

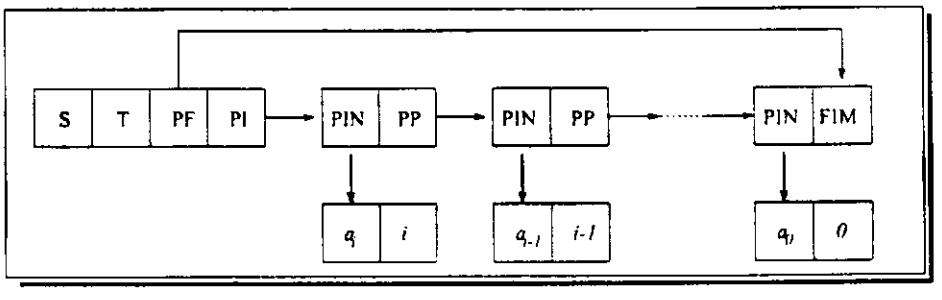

Figura 4.4: Lista para armazenar Inteiros e Polinômios de uma variável

Polinômios de Várias Variáveis

Lista encadeada, com os seguintes elementos, estruturados como na figura 4.5:

- Cabeça da lista, contendo o tamanho da lista principal, o número de variáveis e o grau do polinômio, apontador para o início da lista e apontador para o final da lista.

- Nó principal da lista, contendo o sinal de coeficiente, o tamanho de lista dos coeficientes, o apontador para o início da lista dos coeficientes, o apontador para o final da lista dos coeficientes, a lista de expoentes e o apontador para o próximo nó.

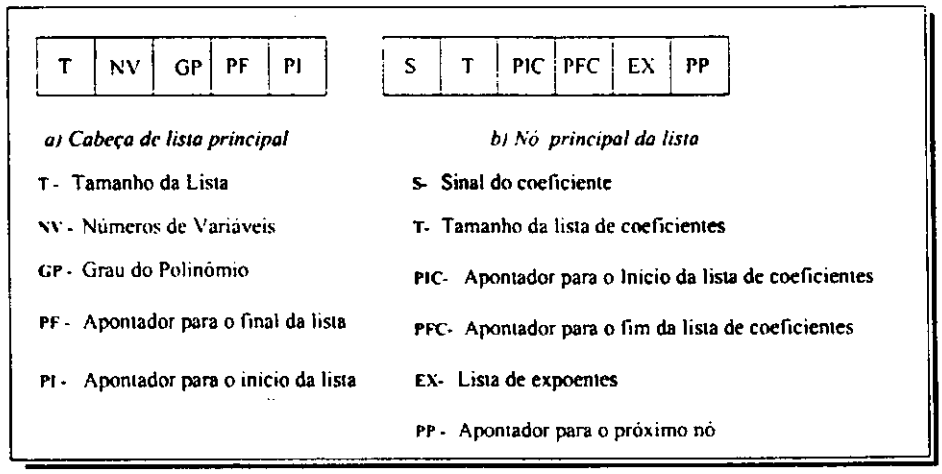

Figura 4.5: Elementos da lista para o armazenamento de polinômios de várias variáveis

Cada nó principal tem apontadores para o primeiro e o último nó da lista que contém o coeficiente de cada termo do polinômio, ficando disposto como na figura 4.6.

O elemento EX da lista apresentada na figura 4.5 pode ser representado de duas formas:

- Se os expoentes forem numéricos o EX deve apontar para uma lista que conterá ordenadamente os expoentes de cada variável (ver Figura 4.7).

- Se os expoentes forem expressões algébricas o EX deve apontar para uma lista que por sua vez terá apontadores para as estruturas em árvores.

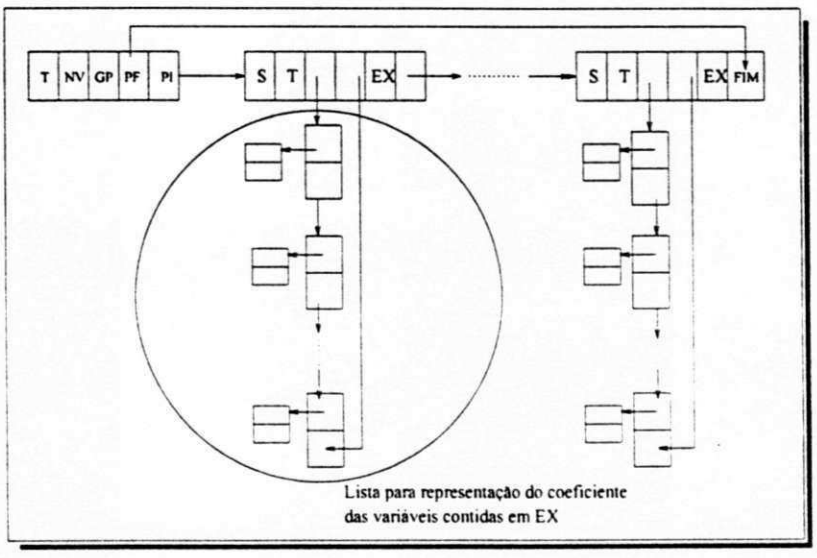

Figura 4.6: Lista de listas para o armazenamento de polinômios de várias variáveis

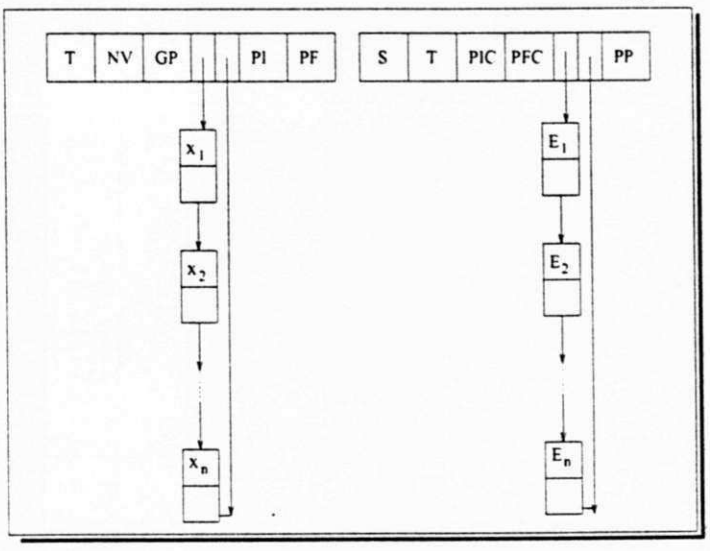

Figura 4.7: Armazenamento dos expoentes para polinômios de várias variáveis

A ordem das variáveis poderá ser apresentada no cabeçalho que teria um apontador para uma lista cuja função seria armazenar as variáveis. Na mesma ordem, seriam armazenados os expoentes a partir de uma lista apontada pelo EX do Nó dado.

Polinômios com coeficientes e expoentes generalizados

Para polinômios cujos coeficientes e expoentes são expressões algébricas, a representação exige um pouco mais de sutileza. O Nó da lista deve conter um apontador para uma estrutura em árvore, que represente o coeficiente. O EX por sua vez apontará para uma lista que ordenadamente apontará para estruturas em ávore, ou nós de lista, dependendo dos coeficientes serem expressões algébricas ou numéricas, respectivamente.

Pavelle [11] sugere representações para polinômios de varias variáveis, caso sejam considerados como foi citado no programa PM [35], e para funções algébricas. Na figura 4.8 é ilustrada essa representação.

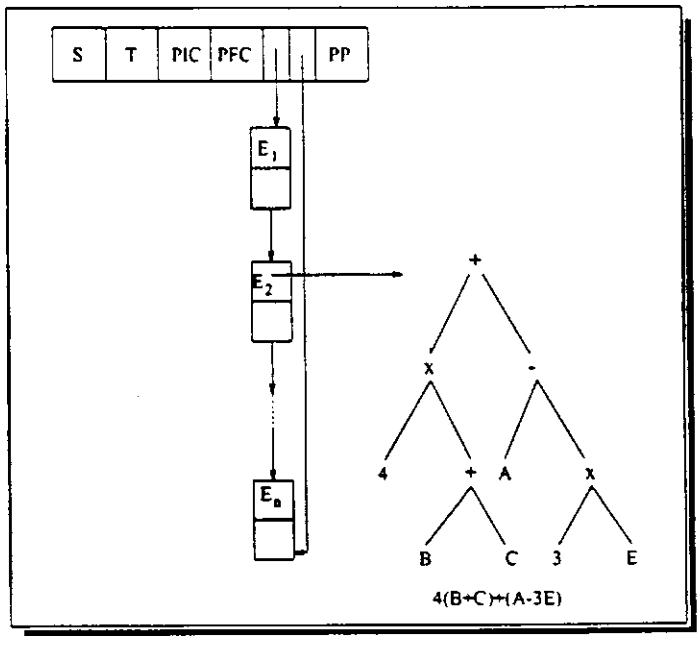

Figura 4.8: Armazenamento dos expoentes para polinômios de várias variáveis

4.9 Números Reais

Na matemática supõe-se que os números reais (Re) formam um conjunto em que são definidas algumas operações e uma relação de ordem. Tanto a relação quanto as operações satisfazem alguns axiomas ou postulados a partir dos quais outras propriedades são deduzidas.

O axioma da continuidade implica que existem alguns números reais, cuja representação em alguma base B, requer um número infinito de dígitos. Por exemplo, $\sqrt{2}$ é um número real, mas não tem representação decimal finita. Num computador a representação dos números reais só é possível com um número finito de dígitos. No caso dos reais essa representação viola o axioma da continuidade, gerando erros e violando outros axiomas.

O números reais são representados no computador imitando a notação científica $a \times B^{\epsilon}$, armazenando a e ϵ, como mostra a figura 4.9. O s é o sinal de a. O a e o e são representados nas bases B e B' respectivamente, em que B' não é necesariamente é igual a B. a é um número real com vírgula decimal implícita à esquerda ou à direita e ϵ é um inteiro relativo.

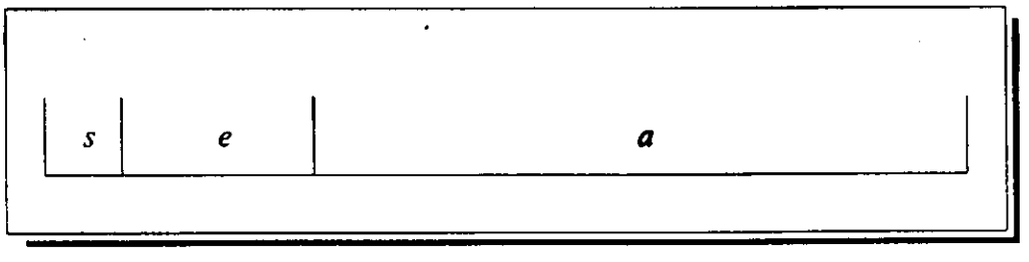

Figura 4.9: Representação de reais em ponto flutuante

Essa maneira de representar os números reais chama-se **representação em ponto flutuante** e constitui um modelo real do conjunto de números reais (abstrato). A informação a é conhecida como **mantissa** e ϵ como o **expoente**. Nem todo número real pode ser representado em ponto flutuante, pois a quantidade de dígitos que pode ser armazenada em a e ϵ é finita.

O conjunto de números em ponto flutuante é caracterizado pelas grandezas, B, m, M, t, em que:

- B é a base utilizada na representação da mantissa;
- t é o número de dígitos, na base B, da mantissa, chamado precisão;
- m é o menor expoente representável na base B';
- M é o maior expoente representável na base B'.

Assim um **sistema de ponto flutuante** pode ser identificado usando a notação $PF(B, t, m, M)$, cujos parâmetros B, t, m, M são dependentes do hardware de um sistema de computação.

Seja $PF(B, t, m, M)$ um sistema de ponto flutuante, e F o conjunto de números reais que pode ser representado nesse sistema. A diferença fundamental entre executar operações aritméticas em F e em Re está no fato de que em F as operações não são fechadas, isto é, se n_1 e n_2 estiverem em F, $n_1 + n_2$, $n_1 - n_2$, $n_1 \times n_2$ e n_1/n_2 podem não pertencer a F, o que ocorre nos seguintes casos:

1. $|n| > Q = (1 - B^{-t})B^M$: neste caso $e > M$ e ocorre o que se conhece como overflow (transbordamento);
2. $0 < |n| < q = B^{m-1}$: neste caso $e < m$ e ocorre o que se conhece como underflow (termo intraduzível)
3. $q \leq |n| \leq Q$, mas a representação de n na base B requer mais de t dígitos na mantissa.

A situação 3 é remediada aproximando n por $n' \in F$.

Chama-se **arredondamento** a aproximação de n por n' e a maneira como n' é determinado chama-se **método de arredondamento**. A diferença $n - n'$ chama-se **erro de arredondamento**. Para evitar que um erro de arredondamento cometido na representação de um número real n seja ampliado em operações aritméticas é interessante transformar a representação n' em ponto flutuante para inteiro e então executar operações aritméticas. Isso garante que as operações aritméticas não produzam erros além do cometido na representação de n'.

Se o número real estiver armazenado em ponto flutuante, então ele pode ser convertido para a forma fracionária p/q, $q \neq 0$, e operado nesta forma, para que os erros de arredondamento não sejam ampliados. O REDUCE trabalha com esse formato. Se o número real não estiver armazenado em ponto flutuante ou caso o usuário opte pela forma na notação posicional $(xxxx, fff)$, sugere-se neste trabalho que as operações aritméticas básicas sejam executadas internamente usando aritmética de *ponto fixo*.

Um número real pode ser representado internamente em um computador com erro, quando tem uma representação infinita, ou sem erro, quando a representação for finita. Serão discutidas duas formas de manipular números reais, admitindo que os números fornecidos como operando das operações tenham representação exata.

4.9.1 Notação Racional

Os racionais são quocientes de números inteiros (notação racional) e podem ser manipulados aproveitando as operações definidas para inteiros.

Adotando a notação:

$$N_B = x_{n-1}x_{n-2}...x_1x_0.f_1f_2...f_k = \sum_{i=n}^{0} x_{i-1}B^i + \sum_{i=1}^{k} f_iB^{-i},$$

onde $B > 1$, é possível multiplicar e dividir N_B por B^k, em que k é o número de dígitos depois da vírgula. Por exemplo, se

$$N_B = x_{n-1}x_{n-2}...x_1x_0.f_1f_2f_3,$$

então é possível multiplicar e dividir por B^3 para obter,

$$N_B = x_{n-1}x_{n-2}...x_1x_0.f_1f_2f_3 \times \frac{B^3}{B^3} = \frac{x_{n-1}x_{n-2}...x_1x_0f_1f_2...f_3}{B^3}.$$

Com isso o número real na notação posicional passa a ser escrito na notação racional, (p/q), e com ele executar as operações aritméticas.

4.9.2 Aritmética de Ponto Fixo

Se um usuário fornecer números reais na notação posicional, espera obter resultados exatos de operações aritméticas com esses números. Um resultado exato significa todos os dígitos corretos, inclusive aqueles da parte fracionária. Como uma mudança de base para representar frações nem sempre é exata, um sistema de computação algébrica não deve usar a representação em ponto flutuante disponível no Hardware. Para evitar conversão de frações o sistema deve dar um tratamento especial aos números reais fornecidos pelos usuário na notação posicional. Esse tratamento usa uma representação especial chamada *ponto fixo*, que permite administrar exatamente a parte fracionária resultante de operações aritméticas, exeto no caso de uma operação de divisão não exata.

Definição 4.9.1 *Dado um número real qualquer N, ele pode ser representado pela tripla $(D; f; S)$, onde D é uma n-úpla com os dígitos de N, f é o número de dígitos depois da vírgula, e S o sinal do número real N, essa representação chama-se representação em ponto fixo.*□

Exemplo 4.9.1 Para

$$N = x_{n-1} \cdots x_3 x_2 x_1 x_0 , f_1 f_2 f_3 \cdots f_m ,$$

na notação $(D; f; S)$, tem-se :

$$D = x_{n-1} \cdots x_3 x_2 x_1 x_0 f_1 f_2 f_3 \cdots f_m \ ; \ \mathrm{f} = \mathrm{m} \ \mathrm{e} \ \mathrm{S} = +.$$

Definição 4.9.2 *O comprimento de N, denotado por $comp(N)$, é o número de dígitos de D.* □ .

No exemplo acima $comp(N) = n + m$.

Dados dois números $N_1 = (D_1; f_1; S_1)$ e $N_2 = (D_2; f_2; S_2)$ representados em ponto fixo, para obter a soma, a diferença, o produto e o quociente desses números, D_1 e D_2 são considerados inteiros positivos e ai é possível usar os algoritmos que executam as quatro operações básicas da seção 4.1. Para isso o alinhamento das vírgulas para D_1 e D_2 é feito na base 10, para então converter esses números para base $B \leq 2^{\frac{k}{2}-1}$

e armazenar em listas encadeadas, para executar operações aritméticas. O resultado da operação será convertido para base 10. Isso evitará a introdução de novos erros de arredondamento devido a conversão de base.

Adição e Subtração

Sejam $N_1 = (D_1; f_1; S_1)$ e $N_2 = (D_2; f_2; S_2)$ dois números reais em sua representação em ponto fixo. Para executar a adição ou subtração entre N_1 e N_2 e obter $N_A = (D_A; f_A; S_A)$ e $N_S = (D_S; f_S; S_S)$ respectivamente, deve-se alinhar as vírgulas, de acordo com as regras abaixo.

- Se $f_1 > f_2$, então $D_2 := D_2 \times 10^k$, onde $k = f_1 - f_2$ e $f_A = f_1$ ou $f_S = f_1$.
- Se $f_1 < f_2$, então $D_1 := D_1 \times 10^k$, onde $k = f_2 - f_1$ e $f_A = f_2$ ou $f_S = f_2$.
- Se $f_1 = f_2$, então as vírgulas já estão alinhadas e $f_A = f_2$ ou $f_S = f_2$.

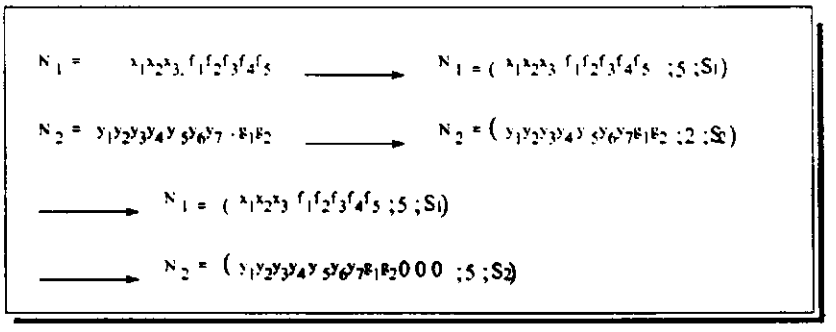

Figura 4.10: Alinhamento das vírgulas para operações de adição e subtração com números reais

Na figura 4.10 N_1 é um número real com 5 casas decimais e N_2 é um número real com 2 casas decimais, depois de comparado os f's, o D_2 é multiplicado por 10^3 e as vírgulas são alinhadas. Então $f_1 = f_2 = 5$.

A base utilizada é a base decimal razão porque, para o alinhamento multiplica-se por 10^k.

Alinhadas as vírgulas, é necessário saber qual a operação a ser efetivamente executada. Para isso é feito um estudo na relação dos sinais de N_1 e N_2.

Será adotada a notação $S_1 \times S_2$ para indicar o sinal resultante numa multiplicação ou divisão, tendo valor + se $S_1 = S_2$ e valor − se $S_1 \neq S_2$.

- Adição - O usuário deseja obter $N_1 + N_2$
  - Se $S_1 \times S_2 = +$, tem-se $S_1 = S_2$, então $D_A := D_1 + D_2$ e $S_A = S_1$.
  - Se $S_1 \times S_2 = -$, tem-se $S_1 \neq S_2$, então $D_A := |D_1 - D_2|$ e outro teste deverá ser feito:
    * Se $D_1 > D_2$ então $S_A = S_1$.
    * Se $D_1 < D_2$ então $S_A = S_2$.
- Subtração - O usuário deseja obter $N_1 - N_2$
  - Se $S_1 \times (-S_2) = +$, tem-se $S_1 \neq S_2$.então $D_S := D_1 + D_2$ e $S_A = S_1$.
  - Se $S_1 \times (-S_2) = -$, tem-se $S_1 = S_2$.então $D_S := |D_1 - D_2|$ e outro teste deverá ser feito:
    * Se $D_1 > D_2$ então $S_S = S_1$.
    * Se $D_1 < D_2$ então $S_S = -S_2$.

O teste dos sinais descrito acima, indica qual a operação a ser efetivamente executada, ficando a determinação da operação efetiva transparente ao usuário. Quando o usário desejar obter $N_1 + N_2$, D_A será indicado como Soma(D_1, D_2), e quando o usuário desejar obter $N_1 - N_2$, D_S será indicado como Diferença(D_1, D_2),

Exemplo 4.9.2

Para se obter $N_1 + N_2 = N_A = (D_A; f_A; S_A)$, com $N_1 = (D_1, 3, +)$ e $N_2 = (D_2; 2; +)$, devem ser executados os seguintes passos:

1. Teste de alinhamento das vírgulas: Como $f_1 > f_2$, calcula-se $k := f_1 - f_2 = 3 - 2 = 1$, $D_2 := D_2 \times 10$ e adota-se $f_A := f_1$.
2. Calcula $D_A := D_1 + D_2$ e $S_A := S_1$
3. $N_A = (D_1 + D_2; f_1; S_1)$ □

Exemplo 4.9.3

Para obter $N_1 - N_2 = N_S = (D_S; f_S; S_S)$, com $N_1 = (D_1, 3, +)$ e $N_2 = (D_2; 2; +)$, devem ser executados os seguintes passos:

1. Teste de alinhamento das vírgulas; Como $f_1 > f_2$, calcula-se $k := f_1 - f_2 = 3 - 2 = 1$, $D_2 := D_2 \times 10$ e $f_A := f_1$.
2. Calcula $D_S := |D_1 - D_2|$
3. Como $D_1 < D_2$ então $S_S := -S_2$
4. $N_S = (|D_1 - D_2|; f_1; -S_2)$ □

Assim, na notação de ponto fixo é possível representar a adição e a subtração como:

$$\begin{aligned} N_1 + N_2 &= (\text{Soma}(D_1, D_2); \max\{f_1, f_2\}; S_A) \\ N_1 - N_2 &= (\text{Diferença}(D_1, D_2); \max\{f_1, f_2\}; S_S), \end{aligned}$$

Multiplicação

A multiplicação é a operação mais fácil de ser executada. Para os números reais em ponto fixo $N_1 = (D_1; f_1; S_1)$ e $N_2 = (D_2; f_2; S_2)$, o produto $N_1 \times N_2 = N_P = (D_P; f_P; S_P)$, pode ser obtido da seguinte forma:

$D_P := D_1 \times D_2$, executada usando um dos algoritmos de multiplicação da seção 4.1; $f_P := f_1 + f_2$; e $S_P = S_1 \times S_2$

Assim:

$$N_P = N_1 \times N_2 = (D_1 \times D_2; f_1 + f_2; S_1 \times S_2)$$

Divisão

Dados $N_1 = (D_1; f_1; S_1)$ e $N_2 = (D_2; f_2; S_2)$ deseja-se encontrar, $N_1/N_2 = N_D = (D_D; f_D; S_D)$, com $D_D \approx D_1/D_2$, em que D_1 ou D_2 é ajustado de acordo com a regra abaixo deixando N_1 e N_2 com o mesmo número de dígitos na parte fracionária.

- Se $f_1 > f_2$, calcula-se $k = f_1 - f_2$, $D_1 := D_1 \times 10^k$.
- Se $f_1 < f_2$, calcula-se $k = f_2 - f_1$, $D_2 := D_2 \times 10^k$.
- Se $f_1 = f_2$, então os dígitos de D_1 e D_2 já estão ajustados.

É possível entender a utilidade da regra acima observando os seguintes exemplos:

Exemplo 4.9.4

(a) Para $B = 10$

$$N_1 = 7 = (7:0:+), \quad D_1 = 700 = 7 \times 10^2.$$
$$N_2 = 0.14 = (14:2:+).$$

O valor do quociente obtido na divisão 700/14 é igual ao valor obtido na divisão 7/0.14.

(b) Para $B = 10$

$$N_1 = 12.45 = (1245:2:+).$$
$$N_2 = 8.3 = (83:1:+) \quad D_2 = 83 \times 10.$$

O valor do quociente obtido na divisão 1245/830 é igual ao valor obtido na divisão 12.45/8.3.□

A divisão de dois números reais pode ser exata ou não. Muitas vezes se a divisão prosseguir com mais dígitos que os fornecidos pelos usuários é possível capturar todos os dígitos do quociente. Quando 7.5 é dividido por 2, é possível obter a resposta exata prosseguindo com a divisão além das casas decimais forcecidas pelo usuário.

Devido a esse problema, arbitrou-se $f = \max\{f_{mim}, f_1 + f_2\}$, desta forma é possível fornecer ao usário uma resposta com no mínimo f_{mim} dígitos na parte fracionária.

Calcula-se $D_1 := D_1 \times 10^{f+1}$, para permitir que o quociente seja obtido com um dígito a mais que a precisão desejada, a fim de possibilitar um arredondamento.

Observe que D_1 e D_2 estão na base 10. Para usar o algoritmo da divisão da seção 4.1, D_1 e D_2 devem ser convertidos para a base $B \leq 2^{\frac{k}{2}-1}$, obtendo um quociente Q_B na base B.

O algoritmo da divisão calcula o quociente de dois números inteiros dígito a dígito. Assim o último dígito de Q_B é examinado para aplicar a regra de arredondamento para

o mais próximo[2], antes de ser descartado. Depois do arredondamento Q_B é convertido para base 10 (Q_{10}).

Assim,

$$N_D = \frac{N_1}{N_2} = (Q_{10}; f; S_1 \times S_2).$$

Os exemplos seguintes ilustram os procedimentos para obter o quociente de dois números reais utilizando aritmética de ponto fixo. Para facilitar o entendimento, nos exemplos, adimite-se que todo o processo de divisão ocorre em base 10.

Exemplo 4.9.5

Para $B = 10$

$$\begin{aligned} N_1 &= 7.5 = (75; 1; +) \\ N_2 &= 2.0 = (2; 0; +) \quad D_2 := 2 \times 10 = 20 \end{aligned}$$

O valor do quociente obtido na divisão 75/20 é igual ao valor obtido na divisão 7.5/2.

Neste caso $f = \max\{3, 1 + 0\} = 3$, $D_1 := 75 \times 10^{f+1} = 750000$.

O valor do quociente $Q = 750000/20$ será 37500. Q é obtido com 5 dígitos, um dígito a mais que a precisão desejada. Usando a regra do arredondamento para o mais próximo encontra-se $Q = 3750$.

[2]Arredondamento é um processo para representar um número com $r + t$ dígitos por uma aproximação de t dígitos. Para isso precisa-se descartar os r dígitos mais à direita. Para minimizar o erro, de arredondamento na representação obtida, o arredondamento para o mais próximo é usado:

- Se e o dígito $t + 1$ for menor que $B/2$, apenas descartam-se os dígitos $t + 1$, $t + 2$, ...
- Quando o dígito $t + 1$ for exatamente $B/2$, depois de descartar os dígitos $t + 1$, $t + 2$, ..., soma-se 1 ou não ao dígito t para que o número obtido seja par.
- Se o dígito $t + 1$ for maior que $B/2$ soma-se 1 ao dígito t, antes de descartar os dígitos $t + 1$, $t + 2$, ...

Para se efetuar um arredondamento para o mais próximo é indispensável pelo menos um dígito extra, além dos t dígitos de precisão, porque precisa-se saber qual é o valor do dígito $t + 1$. Esse dígito extra é chamado dígito de guarda.

Assim,

$$N_D = \frac{N_1}{N_2} = (3750; 3; +) = 3,750\square$$

Exemplo 4.9.6

Para base $B = 10$

$$\begin{aligned} N_1 &= 20 = (20; 0; -) \quad D_1 := 20 \times 10 := 200 \\ N_2 &= 3,3 = (33; 1; +) \end{aligned}$$

O valor do quociente obtido na divisão 200/33 é igual ao valor obtido na divisão 20/3,3.

Neste caso $f = \max\{3,1 + 0\} = 3$, $D_1 := 200 \times 10^{f+1} = 2000000$.

O valor do quociente $Q = 2000000/33$ será 60606. Q é obtido com 5 dígitos, um dígito a mais que a precisão desejada. Usando a regra do arredondamento para o mais próximo encontra-se $Q = 6061$.

Assim,

$$N_D = \frac{N_1}{N_2} \approx (6061,3; -) \approx -6,601\square$$

Exemplo 4.9.7

Para base $B = 10$

$$\begin{aligned} N_1 &= 5,0 = (5; 0; -) \\ N_2 &= 3,0 = (2; 0; +) \end{aligned}$$

Neste caso $f = \max\{3,0 + 0\} = 3$, obtendo $D_1 := 5 \times 10^{f+1} = 50000$.

O valor do quociente $Q = 50000/3$ será 16666. Q é obtido com 5 dígitos, um dígito a mais que a precisão desejada. Usando a regra do arredondamento para o mais próximo encontra-se $Q = 1667$.

Assim,

$$N_D = \frac{N_1}{N_2} \approx (1667, 3; -) \approx -1,667\square$$

Os números transcendentais podem ser manipulados simbolicamente, como e e π. O problema surge quando seus valores numéricos são necessários. Para calcular seus valores numéricos existem muitos algoritmos. Para o cálculo de π, por exemplo, o algoritmo de Borwein e Borwein [40] pode calcular o valor com a precisão de 1.000.000 dígitos. Esses algoritmos podem ser usados nos sistemas algébricos.

4.10 Linguagem de Pogramação

Alguns dos requisitos mais desejáveis de uma linguagem de programação a ser usada na implementação de um sistema de software algébrico estão relacionados a seguir:

1. Deve ser padronizada por instituições internacionalmente reconhecidas como ISO, para que os programas escritos nessa linguagem possam ser compilados e executados em qualquer ambiente de computação que disponha de um compilador para a linguagem. Equivale a exigir que os programas sejam portáteis; essa exigência pode ser impossível de ser satisfeita por todos os programas do sistema, mas a maioria deve satisfazer.

2. Deve permitir trabalhar com um número razoável de tipos de dados e agregados de um ou mais desses tipos. Além disso, deve permitir ao programador definir tipos de dados de sua conveniência.

3. Deve possuir boa estrutura de controle.

4. Deve permitir a criação de funções e procedimentos com regras bem definidas de passagem de parâmetros e sem efeitos colaterais.

5. Deve permitir uma alocação dinâmica de espaço de memória para facilitar a utilização de estruturas de dados dinâmicas.

6. Deve ter recursos usuais para manipular cadeias de caracteres.

Capítulo 5

Algoritmo de Risch

No capítulo 3 foram descritos alguns módulos de um sistema algébrico, entre eles o módulo para obtenção de primitivas. No decorrer da dissertação foi destacada a dificuldade em obter um algoritmo para encontrar a primitiva de uma função. Obter funções primitivas é muito mais difícil do que obter funções derivadas. Para obtenção de funções derivadas, existem procedimentos sistemáticos que usam regras bem definidas. Para obter funções primitivas, entretanto, não existe procedimento sistemático. Existem alguns princípios gerais, mas muitas integrais não podem ser solucionadas usando esses princípios.

A integração simbólica, historicamente, foi um dos grandes desafios da computação algébrica. O algoritmo de Risch, baseado no teorema de Liouville, é discutido neste capítulo a fim de mostrar e compreender os detalhes do seu funcionamento, sugerindo a sua possível implementação. Também será apresentada uma visão geral de como alguns sistemas tratam a integração.

5.1 Terminologia

Antes de iniciar o capítulo é conveniente definir alguns termos.

Definição 5.1.1 : *Seja $F(x)$ tal que a sua derivada $F'(x) = f(x)$. Então $F(x)$ é uma primitiva de $f(x)$.* □

Definição 5.1.2 : *A operação inversa da diferenciação é a integração, que consiste em dada $f(x)$, obter $F(x)$ tal que $F(x)$ seja a primitiva de $f(x)$.* □

Aqui será usado o termo integração ou integração simbólica para a operação que obtém a primitiva de uma função.

Definição 5.1.3 : *Dizer que* $\int f(x)dx = F(x)+c$ *em* I, *onde* I *é um subconjunto dos reais, (ou para* x *em* I*) significa que* F *é uma antiderivada (primitiva) de* f *em* I. *Por causa da natureza arbitrária da função* c, $\int f(x)dx$ *é chamada integral indefinida.* □

Para verificar uma afirmativa da forma $\int f(x)dx = F(x) + c$, é necessário apenas verificar que $F'(x) = f(x)$ para todos os valores no domínio de f .

Definição 5.1.4 : *Seja* f *uma função definida no intervalo fechado* $[a, b]$. *A expressão* $\int_b^a f(x)dx$ *é chamada integral definida de* a *até* b *de* $f(x)$. *Se* f *for uma função contínua sobre o intervalo fechado* $[a, b]$ *e se* $\int f(x)dx = F(x)+c$, *então* $\int_b^a f(x)dx = F(b)-F(a)$. □

Para $\int f(x)dx$ (definida ou indefinida), $f(x)$ é chamado integrando. O termo integral se refere ao resultado da integração definida.

Observe que ao se calcular $\int f(x)dx$ obtém-se uma função, a *primitiva* de $f(x)$, enquanto que ao calcular $\int_b^a f(x)dx$ obtém-se um número como resposta.

A relação entre integral indefinida e definida é a seguinte: Se $F(x)$ for a primitiva de $\int f(x)dx$ então $\int_b^a f(x)dx = F(b) - F(a)$.

5.2 Integração Simbólica

Até o início da década de 60 apenas seres humanos eram capazes de encontrar a função primitiva numa integração indefinida. A solução do problema de integração é um pouco ambígua, razão porque a avaliação da integral indefinida por computador era considerada até então inviável devido a necessidade de heurística, conhecimento e inteligência para determiná-la. No entanto, por volta do final da década existiam programas mais rápidos e por vezes superando a capacidade dos seres humanos. Nos dias de hoje, os avanços permitiram, finalmente, que se implementassem programas com muito mais capacidade que qualquer ser humano [41].

Três principais abordagens na integração simbólica podem ser destacadas na década de 60 [13]:

1. A abordagem da Inteligência Artificial.
2. Abordagem da Manipulação Algébrica.
3. O tratamento Matemático.

A seguir será dado maiores detalhes sobre estas abordagens.

5.2.1 A Abordagem da Inteligência Artificial

Um representante desta abordagem é o programa pioneiro SAINT (Symbolic Automatic INTegrator) escrito por Slagle [42] em liguagem LISP, que fazia uso de técnicas correntes de simplificação de integrais e de uma tabela de integrais básicas. O outro é programa SIN (Symbolic INtegrator) escrito por Moses [43] posteriormente incorporado aos sistemas gerais de manipulação algébrica MACSYMA e SCRATCHPAD, também escrito em LISP, contém além dos métodos tradicionais de solução, uma versão do algoritmo de Risch para integração de funções elementares (será definida posteriormente).

Pode-se pensar num programa para integração como sendo um "programa para solução de problemas". Tem-se um problema bem definido e uma variedade de métodos de solução, alguns mais eficientes que outros. Deve-se implementar um programa que vise dois objetivos básicos:

(i) Encontrar a solução do problema de obtenção de uma primitiva de uma função de uma maneira eficiente.

(ii) Obter a solução cuja forma seja *semelhante* à forma do integrando, isto é, que contenha funções presentes no integrando.

A estratégia utilizada pelo programa SIN ilustra a abordagem adotada. O problema é tratado em três estágios:

I. Procura resolver o problema proposto por um método geral e pouco dispendioso, que consiste em verificar se a integral dada é da forma:

$$\int kf(u(x))u'(x)dx,$$

em que,

k é uma constante;
$u(x)$ é uma função qualquer de x;
$u'(x)$ é a derivada da função $u(x)$ em relação a x;
$f(x)$ é uma função elementar, podendo ainda representar $u(x)$, $u(x)^d$ e $d^{u(x)}$, em que d é uma constante.

O sistema deve ser capaz de comparar e relacionar o integrando do problema com um integrando disponível em uma tabela de funções; se forem iguais substitui-se x por $u(x)$ na expressão da tabela.

II. Este estágio contém onze métodos para solucionar o problema de integração e que podem ou não ser aplicáveis dependendo do tipo de integral proposta. Para isso é embutida uma inteligência no programa que o torna capaz de examinar o integrando e dependendo de sua forma e de que funções apareçam em sua expressão. escolhe um método a ser aplicado.

III. Se não for possível resolver por nenhum desses métodos, aplica-se um método geral de solução. Neste estágio é aplicado um procedimento que é um subconjunto do algoritmo de Risch, já mencionado e ao qual será dada uma maior atenção em uma seção posterior.

5.2.2 Abordagem da Manipulação Algébrica

Um representante desta abordagem é o programa MATHLAB desenvolvido por Manove, Bloom e Engenman [44] para integração de funções racionais. Os trabalhos teóricos de Tobey [41] e Horowitz [45] serviram de base para o programa MATHLAB.

Em programas de manipulação algébrica é enfatizada a necessidade de grande eficiência nas operações relativamente comuns. O algoritmo de integração de funções racionais (razões entre polinômios) pode ser considerado como o mais comum dentre os algoritmos não triviais para integração. Embora o algoritmo não apresente maiores dificuldades matemáticas, sua implementação em computador apresenta consideráveis dificuldades. Inicialmente, foi construída uma versão desse algoritmo no sistema MATHLAB. Esse algoritmo é o mais poderoso dos métodos do estágio dois do programa SIN.

5.2.3 O Tratamento Matemático

O tratamento matemático é representado pelos trabalhos de Richardson [46] sobre a indecidibilidade da integração de certas classes de funções e pelos trabalhos de Risch [47] a respeito de procedimentos para determinação de funções elementares.

A busca de resultados gerais sobre o problema de integração tem origem no século XIX. Laplace lançou a conjectura de que a primitiva de uma função algébrica (ver definição no Apêndice) contém apenas as funções algébricas presentes no integrando. Mais tarde a validade dessa conjectura foi provada por Abel. Liouville examinou a forma da integral de uma função elementar (ver definição 5.3.1) numa série de trabalhos publicados entre 1830 e 1840 [13]. O teorema Principal de Liouville foi a base para a maior parte dos trabalhos recentes em integração.

Os trabalhos de Richardson e principalmente de os Risch levaram à concepção de um algoritmo, baseado no teorema de Liouville, que verifica se uma dada integral de função elementar pode ser expressa como combinação de funções elementares e constrói a solução em caso afirmativo.

5.3 Teorema de Liouville

É possível uma primitiva de uma função de uma variável sempre ser obtida em sua forma fechada ou em um número finitos de termos? Quem estabeleceu regras para obter primitivas em forma de uma expressão finita foi Joseph Liouville cujos trabalhos nesse assunto foram publicados no período 1833-1841 [47]. O teorema de Liouville em sua forma simples afirma que se uma função algébrica (ver definição no Apêndice) tiver como primitiva uma função elementar, então uma primitiva é uma função algébrica adicionada a uma combinação de logaritmos de funções algébricas. Ostrowski generalizou o Teorema de Liouville para uma ampla classe de funções meromórficas (ver definição 5.3.2) definidas em regiões do plano complexo e J. F. Ritt trata desse assunto em [48].

Para uma melhor comprensão deste capítulo é preciso estar familiarizado com conhecimentos de álgebra abstrata e de análise complexa. Alguns conceitos básicos e referências sobre o assunto são fornecidos no Apêndice, cuja leitura é recomendada. Alguns termos serão definidos a seguir.

Definição 5.3.1 *[47]: Uma função elementar é usada aqui para designar funções construídas a partir de funções racionais usando apenas funções exponenciais, logarítmicas,*

trigonométricas, trigonométricas inversas e operações algébricas (adição, subtração, multiplicação e divisão).□

A primitiva de uma função racional de uma variável real é elementar se ela for uma combinação linear de logaritmos, função inversa da função tangente e funções racionais.

Definição 5.3.2 *[49]: Uma função meromórfica sobre uma região do plano complexo é uma função cujos valores são números complexos, com a propriedade de que suficientemente perto de qualquer ponto z_0 da região do plano complexo, a função é dada por uma série de Laurent em $z - z_0$, isto é, uma série de potências convergente em $z - z_0$, com a possível adição de um número finito de números de potência negativa.*□

Definição 5.3.3 *[49]: Um anel diferencial é um anel comutativo R (ver definição no Apêndice) junto com uma operação unária (opera com apenas um elemento do domínio) de derivação definida em R com valores em R, isto é, uma função de $R \to R$ denotada por $u \longmapsto u'$ satisfazendo duas regras:*

a) $(u + v)' = u' + v'$

b) $(uv)' = u'v + v'u$ □

Um *corpo diferencial* é um anel diferencial que é um corpo. Se u e v forem elementos de um corpo diferencial e $v \neq 0$ então $(u/v)' = (u'v - uv')/v^2$. Em um anel, $(u^n)' = nu^{n-1}u'$ para $n = 1, 2, 3...$. Em particular, se $u = 1$ e $n = 2$ então $u' = 0$. Elementos cujas derivadas são nulas são chamados de constantes, e a totalidade das constantes constitui o subcorpo das constantes.

Se u e v forem elementos de um corpo diferencial tal que $v \neq 0$ e $u' = v'/v$, então u é um logaritmo de v e v é uma exponencial de u, em que u e v são únicos a menos de uma constante.

Definição 5.3.4 *[49]: Uma extensão diferencial de corpos de um corpo diferencial é elementar se existir uma torre (ver definção no Apêndice) finita de corpos diferenciais intermediários, tal que cada corpo seguinte depois do primeiro, seja obtido do seu predecessor pela junção de um elemento que seja logaritmo ou exponencial de um elemento sobre o corpo anterior.*□

Teorema 5.3.1 *(Teorema de Liouville)[49]: Seja $\mathcal{F}$ um corpo diferencial com um subcorpo algebricamente fechado (ver definição no Apêndice) $\mathcal{K}$. Seja $f \in \mathcal{F}$ e g elementar sobre $\mathcal{F}$ com $g' = f$. Então existem $v_0, v_1, ..., v_k$ em $\mathcal{F}$ e $c_1, c_2, ..., c_k$ em $\mathcal{K}$ tal que:*

$$f = v_0' + \sum c_i \frac{v_i'}{v_i}. \square$$

Em outras palavras, a integral de uma função em um corpo de funções elementares é um membro do corpo somado a uma combinação linear de logaritmos de elementos do corpo.

A generalização do teorema de Liouville por Ostrowski está contida no seguinte resultado [48]:

Teorema 5.3.2 *[48]: Sejam $\mathcal{F}$ um corpo diferencial de característica zero (ver definição no Apêndice), e $a \in \mathcal{F}$. Se a equação $y' = a$ tiver uma solução em alguma extensão diferencial de $\mathcal{F}$ tendo o mesmo subcorpo de constantes, então, existem constantes c_1, c_2, ... $c_n \in \mathcal{F}$ e elementos $u_0, u_1, ..., u_k, v$ em $\mathcal{F}$ tal que:*

$$a = \sum c_i \frac{u_i'}{u_i} + v. \square$$

5.4 Primitiva de uma Função na Forma de uma Expressão Finita

Dada uma função integrando constituída por uma combinação de outras funções, a primitiva pode conter funções que não aparecem na composição da função integrando. Daí, o porquê dos matemáticos considerarem o problema de integração indefinida tão difícil de solucionar. Exemplos de primitivas que não contém o integrando são $\int \sin x dx = -\cos x$ e $\int \left(\sqrt{1-x^2}\right)^{-1} = \sin^{-1} x$.

Algoritmos para resolver um problema de integração indefinida vêm despertando o interesse dos matemáticos por três séculos. Eles buscavam um algoritmo para decidir quando uma função elementar tem uma integral indefinida e encontrá-la, caso exista. Até 1905 pensava-se que todos problemas de integração eram insolúveis por algoritmos.

Hard um famoso matemático, provou que o problema não tinha solução para um subconjunto de funções elementares. R. Rich [47, 49, 50] provou em 1970 que a conjectura de Hardy era falsa. Risch obteve um algoritmo para determinar quando uma integral de uma função elementar existe em forma fechada e encontrá-la caso exista [?].

Deseja-se um algoritmo que transforme um integrando (entrada) em uma primitiva (saída). Então deve-se contar com a existência de diferentes funções que aparecem na primitiva mas que não aparecem no integrando. Às vezes há uma grande diferença entre as funções do integrando e da primitiva, que possivelmente estão ligadas por alguma relação matemática. Para os exemplos citados no início desta seção, têm-se:

$$\sin^2 + \cos^2 = 1 \text{ e}$$
$$\sin^{-1} x = i \log\left(ix + \sqrt{1 - x^2}\right).$$

A idéia básica é expressar o integrando e a primitiva em termos de funções algebricamente independentes, isto é, funções que não possuem uma relação polinomial não trivial entre elas. A utilidade das funções algebricamente independentes está ligada ao desenvolvimento do corpo diferencial. Aproximações modernas de integrais indefinidas usam o artifício da algébra diferencial.

Na álgebra diferencial as funções racionais em x, $R(x)$, formam um corpo fechado sob diferenciação. Um corpo diferencial, $R(x, e^x)$ que contém funções racionais em e^x cujos coeficientes são polinômios ou funções racionais é uma extensão do corpo $R(x)$. Note que $e^{2x} \in R(x, e^x)$; basta observar que $e^{2x} = (e^x)^2$. Por outro lado $e^{x^2} \notin R(x, e^x)$. $R(x, e^x, e^{x^2})$ é uma extensão não trivial de $R(x, e^x)$. É possível representar

$$\frac{x^2 \log(\log x + e^x)}{e^{2x} - \log(x + 1)},$$

como um membro de $R(x, \log x, \log(x+1), e^x, \log(\log x + e^x))$. Note que as funções elementares podem ser consideradas como extensão do corpo de algumas funções racionais, em que cada extensão é formada a partir de um novo logaritmo, exponencial ou função algébrica.

O problema de como obter a primitiva de uma função na forma de uma expressão finita Se $f(x) \in R(x, h_1(x), \ldots, h_k(x))$ for uma extensão do corpo dos racionais, onde os $h_i(x)$ são logaritmos, exponenciais e funções algébricas e se $f(x)$ tiver uma primitiva $g(x)$ que seja uma função elementar, então qual o tipo de extensões

necessária para obter uma extensão de corpos que contenha $g(x)$? A resposta: apenas extensões logarítmicas são necessárias; é conseqüência do Teorema de Liouville, já discutido na seção anterior.

5.5 Algoritmo de Risch

Seja $\mathcal{D}$ um corpo diferencial de característica zero (ver definição no Apêndice), que possui um subcorpo $\mathcal{K}$, chamado de corpo de constante de $\mathcal{D}$, ou seja, $\mathcal{K}$ é o conjunto de todos os $a \in \mathcal{D}$ tal que $a' = 0$.

Definição 5.5.1 *[49]: Se $\mathcal{D}$ for um subcorpo diferencial de $\mathcal{F}$, então $\mathcal{F}$ (e qualquer $f \in \mathcal{F}$) é elementar sobre $\mathcal{D}$ se $\mathcal{F}=\mathcal{D}(\theta_1, \ldots, \theta_n)$, em que cada θ_i satisfaz pelo menos uma das condições que se segue:*

1. *θ_i algébrico sobre $\mathcal{D}$ $(\theta_1, \ldots, \theta_n)$ (ver definição no Apêndice).*
2. *$\theta_i'/\theta_i = f'$ para algum $f \in \mathcal{D}$ $(\theta_1, \ldots, \theta_n)$ (caso de exponencial).*
3. *$f'/f = \theta_i'$ para algum $f \in \mathcal{D}(\theta_1, \ldots, \theta_n)$ (caso de logaritmo).* □

Definição 5.5.2 *[49]: Se na definição 5.4.1, em 2 ou 3, θ_i for transcendental (ver definição no Apêndice) sobre $\mathcal{D}(\theta_1, \ldots, \theta_{i-1})$ e o corpo constante de $\mathcal{D}(\theta_1, \ldots, \theta_{i-1})$ for o mesmo que o de $\mathcal{D}(\theta_1, \ldots, \theta_i)$, então θ_i é um monomial sobre $\mathcal{D}(\theta_1, \ldots, \theta_{i-1})$. Se cada θ_i for também algébrico ou monomial sobre $\mathcal{D}(\theta_1, \ldots, \theta_{i-1})$ então $\mathcal{F}$ é uma função é elementar regular sobre $\mathcal{D}$.*□

5.5.1 A Idéia Básica do Algoritmo

Dado um corpo diferencial do tipo $\mathcal{F} = (z, \theta_1, \theta_2, \ldots, \theta_n)$ em que $\mathcal{K}$ é o corpo das constantes de $\mathcal{F}$, $\mathcal{K}$ algebricamente fechado e transcendental de grau finito sobre $\mathcal{Q}$ (conjunto do Racionais). Se $z' = 1$ com $\mathcal{F}$ elementar sobre $\mathcal{K}(z)$, o algoritmo de integração fornece um método para determinar quando uma dada função $f \in \mathcal{F}$ pode ser representada na forma $f = v_0' + \sum c_i \frac{v_i'}{v_i}$, para v_i's em $\mathcal{F}$ e c_i's em $\mathcal{K}$.

O algoritmo prossegue por indução sobre o número de monomiais usados na construção da torre de $\mathcal{K}(z)$ para $\mathcal{F}$. Seja $\mathcal{D}$ elementar sobre $\mathcal{K}(z)$ onde $m-1$ monomiais são usados na torre de $\mathcal{K}(z)$ para $\mathcal{D}$. A hipótese de indução garante que é possível

decidir se os elementos de $\mathcal{D}$ possuem primitivas elementares e também se a equação diferencial linear de primeira ordem com coeficientes em $\mathcal{D}$, com certos parâmetros constantes, tem uma solução em $\mathcal{D}$.

Agora seja θ um monomial sobre $\mathcal{D}$, e ω um elemento algébrico sobre $\mathcal{D}(\theta)$ e $f \in \mathcal{F}$, $\mathcal{F} = \mathcal{D}(\theta, \omega)$. Examina-se a equação $f = v_0' + \sum c_i \frac{v_i'}{v_i}$ em um conjunto finito (determinado por f) na construção de $\mathcal{D}$ para $\mathcal{F}$. Usando a hipótese de indução, certas condições são necessárias para que os $v's$ ocorram de acordo com a representação de Liouville em $\mathcal{F}$, isto é, a parte principal do sistema para v_0 e divisores δ_i, $i = 1, \ldots, k$ tal que (v_i) seja potência de δ_i. O $v's$, se existirem, podem ser determinados por elementos aditivos ou multiplicativos de $\mathcal{D}$. Então o problema é reduzido a estudar a equação $f_1 = d_0' + \sum c_i \frac{d_i'}{d_i}$, com f_1 e d_i's em $\mathcal{D}$.

5.5.2 Desenvolvimento do Algoritmo de Risch

Será apresentada agora, uma idéia de como seria uma possível implementação do algoritmo de Risch.

I. Seja F o conjunto de funções elementares. Inicialmente converte-se o integrando para a forma $P(x)/Q(x)$ em que P e Q são polinômios com coeficientes racionais em x, sendo x um elemento do conjunto F.

Seja N o número de elementos do conjunto F. Pode-se escrever então:

$$F = (x_1, x_2, \ldots, x_n), \tag{5.1}$$

em que os x_i são estruturas do tipo x, e^x, $\log x$, etc.

II. De acordo com o teorema de Liouville pode-se escrever a primitiva a ser encontrada na forma:

$$\int \frac{P(x)}{Q(x)} dx = \frac{U(x_1, x_2, \ldots, x_n)}{Q_p} + \sum_{i=1}^{N_1} c_i \times \log Q_i, \tag{5.2}$$

em que,

U é um polinômio desconhecido;

c_i são constantes correspondentes aos fatores Q_i do tipo exponencial, são nulas.

Nf_i é o número de fatores irredutíveis do polinômio Q;

os Q_i são fatores irredutíveis do polinômio Q ; logo pode-se escrever:

$$Q = \prod_{i=1}^{Nf_i} Q_i; \tag{5.3}$$

o polinômio Q_p é da forma:

$$Q_p = \prod_{j=1}^{Nfe} Q_j^{n_j} \times \prod_{j=1}^{Nfj-Nfe} Q_j^{n_j-1}; \tag{5.4}$$

Na equação (5.4) os fatores Q_j do primeiro produtório são elementos de F do tipo exponencial: Nf_e é número de funções elementares exponenciais; os fatores Q_j do segundo produtório correspondem aos demais fatores irredutíveis de Q ; e n_j é o número de vezes que os fatores Q_j ocorre em Q.

III. Diferenciando a equação (5.2) obtém-se uma equação diferencial linear de primeira ordem no polinômio U.

IV. Considera-se o polinômio U como sendo a soma:

$$U(x_1, x_2, \ldots, x_n) = \sum_{j_N=0}^{INF} \cdots \sum_{j_1=0}^{INF} U_{j_1 \ldots j_n} \times x_1^{j_1} \times x_2^{j_2} \times \ldots \times x_N^{j_N} \tag{5.5}$$

e substituindo a equação (5.5) na equação diferencial obtida no passo **III** pode-se obter relações de recorrência para os coeficientes $U_{j_1 \ldots j_n}$.

V. Determina-se, utilizando as relações de recorrência encontradas, qual o termo da soma (5.5) tem a maior potência $j_1 + \ldots + j_n$. Separa-se o termo correspondente na integral original da equação (5.1) e encontram-se novas relações de recorrência para a parte restante do polinômio.

VI. Repetem-se os passos IV e V reduzindo sucessivamente a potência da parte desconhecida da soma (5.5), escolhendo arbitrariamente c_i, se for necessário.

Finalmente, completa-se o cálculo do polinômio U e dos parâmetros c_i da equação (5.1), isto é, encontra-se a solução. Caso a integral não possa ser colocada na forma (5.1), o Teorema de Liouville garante que ela não pode ser expressa em termos de funções elementares do conjunto F.

A fundamentação matemática se encontra em [47, 51] e exemplos são encontrados em [43]. Alguns exemplos clássicos são apresentados a seguir.

O primeiro exemplo é um problema fácil que pode ser resolvido usando integração por partes. Embora o método do algoritmo de Risch aparente gastar muito mais tempo que o método tradicional, o exemplo ilustra muito bem a idéia usada no algoritmo.

Exemplo 5.5.1

Considere a seguinte integral. $\int \log x dx$.

Claramente $\log x \in R(x, \log(x))$. O integrando é um polinômio em $\log(x)$ de grau 1. Uma vez que a integral deve ser elementar, então ela será no máximo de grau dois.

$$\int \log x dx = B_2(x)\log^2(x) + B_1(x)\log x + B_0(x) + \sum c_i \log V_i(x) \qquad (5.6)$$

em que,

$B_0, B_1, B_2 \in R(x)$.
$V_i(x) \in R(x, \log x)$,
c_i constante.

Note que os B_i's não envolvem $\log x$ e, portanto, estão no corpo $R(x)$.

Diferenciando ambos os membros de (5.6) em relação a x, resulta:

$$\log x = B_2'(x)\log^2(x) + (\frac{2B_2}{x} + B_1')\log x + (\frac{B_1}{x} + B_0') + \sum c_i \frac{V_i'}{V_i}. \qquad (5.7)$$

Como os B_i's são independentes de $\log x$, os coeficientes de $\log^2 x$ e $\log x$ devem ser iguais em ambos os membros da equação.

Assim:

$$B_2'(x) = 0 \quad e \quad \frac{2B_2}{x} + B_1' = 1$$

Integrando ambos os membros de $\frac{2B_2}{x} + B_1' = 1$, e substituindo $2B_2$ por b_2 obtêm-se

$$\int B_1' dx = B_1 + \text{constante},$$
$$\int \frac{b_2}{x} dx = b_2 \log x + \text{constante}.$$

Desta forma, $b_2 \log x + B_1 = x +$ constante. Logo,

$$b_2 = 0 \text{ e } B_1 = x + b_1.$$

Substituindo esses resultados no coeficiente de $(\log x)^0$, tem-se:

$$\begin{aligned} 0 &= \frac{B_1}{x} + B_0' + \sum c_i \frac{v_i'}{v_i}, \\ 0 &= \frac{x + b_1}{x} + B_0' + \sum c_i \frac{v_i'}{v_i}, \\ -1 &= \frac{b_1}{x} + B_0' + \sum c_i \frac{v_i'}{v_i}. \end{aligned}$$

que é a uma forma fechada. Integrando ambos os membros

$$-x + \text{ constante} = b_1 \log x + B_0 + \sum c_i \log v_i.$$

Como b_1 e todos os c_i são nulos, então $B_0 = -x +$ constante. Finalmente,

$$\int \log x dx = x \log x - x + constante. \square$$

Exemplo 5.5.2

Considerere a integral $\int e^{x^2} dx$.

Claramente $e^{x^2} \in R(x, e^{x^2})$. O integrando é um polinômio em e^{x^2} de grau 1. As propriedades de diferenciação da exponencial garantem, portanto, que se a derivada existir como uma função elementar, o grau permanece o mesmo,

$$\int e^{x^2} dx = B_1(x) e^{x^2} + \sum c_i \log v_i(x), \tag{5.8}$$

em que,

$B_1 \in R(x)$.
$v_i(x) \in R(x, e^{x^2})$.
c_i constante.

Diferenciando ambos os membros da equação (5.8) em relação a x:

$$e^{x^2} = (B_1' + 2xB_1)e^{x^2} + \sum c_i \frac{v_i'}{v_i} \tag{5.9}$$

Assim,

$$B_1' + 2xB_1 = 1.$$

Note que B_1 é uma função racional. Se ela tiver um denominador não constante, o grau do denominador de B_1' será maior do que o grau do denominador de B_1, não podendo ser cancelado. Por exemplo, considerando B_1 como sendo um polinômio em x, de grau n, então o grau de B_1' é $n-1$, o grau de $2xB_1$ é $n+1$, o grau da constante 1 é igual a 0. Os termos deveriam ser cancelados, e isso é uma contradição para $n \geq 0$. Logo $\int e^{x^2} dx$ não pode ser expresso em termos de funções elementares, como já é conhecido.□

5.6 Abordagens de alguns Sistemas Algébricos no Tratamento de Integrais

Integração simbólica, historicamente, foi um dos grandes desafios da computação algébrica. A invenção da calculadora e do computador tornou a integração numérica mais fácil para cientistas e engenheiros. Muitos outros estudos encontraram algoritmos para obter primitivas de funções algébricas [52, 53, 54], e de funções que podem ser representadas como funções elementares transcendentais de funções algébricas [55] ou raízes n-ésimas de funções algébricas transcendentais [56].

A seguir são apresentados os recursos de integração oferecidos pelo REDUCE, MAPLE e *Mathematica* [57, 58, 59].

REDUCE

A função INT(EE, X) obtém a integral indefinida da expressão EE em relação a variável X, se for possível. Se não for possível encontrar a solução: 1)Retorna a entrada INT(.....); 2) Retorna a expressão envolvendo INT de alguma outra expressão (às vezes mais complicada que a original).

MAPLE

Calcula integrais definida e indefinida usando a função int, cuja sintaxe pode ser int(f, x), int(f, x= a...b) ou int(f, x=a...b, cont), onde "f" é qualquer expressão algébrica em x, "x" variável independente, "a...b" intervalo e "cont"(opcional) indica que a função é contínua."int" calcula a integral definida ou indefinida de "f" em relação a variável x. A integração indefinida é calculada se o segundo argumento for apenas x. Nenhuma constante de integração aparece no resultado. A integral definida é encontrada se o segundo argumento for "x = a...b", onde "a" e "b" são o limites de integração.

Se não for possível encontrar a primitiva, retorna a função de chamada, ou seja, a função integrando, fornecida pelo usuário.

Uma integral definida de $f(x)$ é encontrada obtendo-se primeiro a primitiva $F(x)$ e depois avaliando $F(b) - F(a)$: caso a primitiva não possa ser encontrada, o valor da integral é obtido numericamente.

Mathematica

É possível integrar qualquer expressão racional (razão entre polinômios). A sintaxe é a seguinte: int[f, x] (integral indefinida), int[f, {x, x_{min}, x_{max}}] (integral definida em relação a x), int[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] (integral definida múltipla).

Divide as integrais em 3 classes: Integrais simples que podem ser resolvidas por métodos simples. Integrais na forma fechada para uma função particular. Integrais cuja primitivas não podem ser expressas usando funções simples ou especiais. Se não for possível encontrar a solução, então a função de chamada é retornada em sua forma simbólica. Uma integral definida de $f(x)$ é encontrada obtendo-se primeiro a primitiva $F(x)$ e depois avaliando $F(a) - F(b)$; caso a primitiva não possa ser encontrada, o valor da integral é obtido numericamente.

É possível adicionar regras para certos tipos de integrais.

Capítulo 6

Conclusões

6.1 Conclusões

Em geral o modelo matemático será resolvido utilizando um método numérico ou um método algébrico. Esses métodos funcionam como uma "sonda" que vai navegando por um espaço desconhecido e que está equipada para enviar informações interessantes sobre um espaço jamais naveagado. A partir destas informações, adquire-se uma intuição mais apurada desse novo "território explorado " e possívelmente fazer descobertas de cunho teórico ou prático. Uma das fases mais importantes de um projeto de matemática computacioanal é a validação. A computação algébrica auxilia oferecerecendo resultados numéricos em menos tempo e obtendo soluções quase sempre exatas, bem como proporcionando discernimento à investigação científica.

O desenvolvimento de softwares para computação algébrica, apresentam muitas dificuldades; A inadequação do hardware disponível é uma delas.

A explosão de operações intermediárias, não permite que as operações básicas sejam executadas por hardware, sendo necessário recorrer a softwares desenvolvidos apenas com esse objetivo. Além disso os periféricos disponíveis para entrada e saída de dados não atendem a necessidade do usuário de software de computação algébrica. A esses problemas é edicionada a dificuldade de desenvolver uma liguagem e uma inerface "universal", isto é, que sejam compreendidas por todos os tipos de usuários do software, independente da aplicação do software e de qual for o nível de interação do usuário ao software.

A solução seria oferecer ao usuário maior grau de liberdade para manipular as expressões, o que implicaria em maiores preocupações com a concepção da interface, com a linguagem a ser utilizada pelo usuário, e acarretaria a necessidade de mais memória. Também seria interessante o desenvolvimento de novos periféricos que facilitassem a entrada e saída de dados.

Como a tendência atual é procurar construir ambientes integrados de solução de problemas (ASP's) de matemática, a cooperação de diversos especialistas é indispensável. Um ASP é um produto que deve incorporar uma tecnologia avançada de computação, a sua produção vai requerer uma gerência sofisticada e a contribuição de muitos especialistas. Possivelmente haverá necesidade de muita pesquisa em quase todas ás áreas de especialização envolvidas antes de emergir uma metodologia para o desenvolvimento de ASP's. A integração de recursos de computação algébrica a um ASP é complicada; parece demandar ainda muito trabalho para se tornar uma tarefa racionalmente tratável.

Pelo menos conceitualmente é possível pensar em um sistema de software algébrico organizado em módulos. Os módulos devem implementar funções bem definidas para que se possa alcançar maior clareza na estrutura do software. É possível executar operações apenas ativando o módulo correspondente à operação desejada. Para isso é necessário embutir alguma inteligência no sistema.

Para resolver o problema de explosão das expressões intermediárias, o sitema de software algébrico deverá conter um módulo (módulo básico) responsável pela execução das operações com inteiros e polinômios. As operações poderão ser executadas em precisão arbitrária, usando listas encadeadas e/ou árvores como estrutura de dados destes objetos.

Os algoritmos para executar as operações básicas utilizam a notação posicional para qualquer base $B > 1$. A base adequada para a execução das quatro operações básicas é $B \leq 2^{\lfloor k/2 \rfloor - 1}$. Essa escolha tem a desvantagem de perder espaço, na representação de um inteiro, mas o produto de dois dígitos pode ser representado em uma palavra e pode ser obtido pela execução da multiplicação disponível em hardware.

Para evitar a propagação dos erros de arredondamento, as operações com números reais podem ser executadas através de software;

1. Utilizando da notação racional.
2. Utilizando a aritmética de ponto fixo. $N = (D; f; S)$:Na notação de ponto fixo, o produto de N_1 e N_2 pode ser obtido sem maiores problemas. A operação de

adição e subtração exige maiores cuidados com o alinhamento das vírgulas e a relação entre os sinais. Para obter o quociente, a multiplicação por pontências de 10 foi utilizada como artifício para capturar os dígitos da parte fracionária do quociente. Isso foi possível porque o algoritmo de a divisão de dois inteiros calcula o quociente dígito a dígito.

3. E manipulando os números transcendentais simbolicamente.

As vantagens destas representações para o número real, é a possíbilidade de aproveitar os algoritmos e as estruturas de dados utilizadas para a manipulação com inteiros.

O problema de integração simbólica merece um tratamento mais cuidadoso em computação algébrica porque para implementá-la é necessária uma certa heurística. Baseado no Teorema de Liouville, o algoritmo de Risch não é difícil de implementar no computador para casos que envolvem apenas funções logaritmicas e exponenciais. O caso em que se trabalha com funções que não são algebricamente independentes exige mais cuidado, necessitando de conhecimento matemático amplo e variado (algébra abstrata, análise complexa, algebra linear, topologia, etc), bem como dispositivos periféricos mais avançados.

Os progressos que redundaram nos computadores eletrônicos atuais, descendentes dos antigos instrumentos de cálculo mecânicos simples, contam entre as facanhas matemática mais importantes do século XX. Particurlamente revolucionária foi a idéia de uma máquina que além de operar com dados armazenasse um programa de instrução. Ao final do século XX, busca-se inserir inteligência em computadores, afim de torná-los cada vez mais independentes da ação humana.

Observa-se que os modelos matemáticos mais recentes são bem mais complexos e exigem um esforço computacional maior. Por isso, a concepção de software matemático torna-se bastante complexa, o que vem exigindo cada vez mais a formação de especialistas que possam conceber produtos necessários à ciência e tecnologia. A necessidade de especialistas já pode ser detectada, em várias áreas como: Engenharia de Software, Inteligência artificial, Computação algébrica e etc.

Pode haver alguns poucos pontos sobre os quais talvez se possa prognosticar um futuro, com alguma segurança:

1. O desenvolvimento da computação eletrônica do século XX deverá prosseguir no futuro por algum tempo, levando a uma velocidade nos cálculos e aplicações difíceis de imaginar hoje.

2. A linha divisória entre a matemática pura e a matemática aplicada deverá se enevoar cada vez mais. Por outro lado é possível entender a matemática toda como uma matemática aplicada, pois a aplicação é, a vezes, uma questão de tempo.

3. A computação algébrica apresentar-se-a como uma feramenta poderosa e indispensável à ciência e tecnologia: principalmente porque a tendencia dos softweres de computação algébrica, utilizados atualmente, é integrar a computação numérica, computação algébrica e a computação gráfica. Alguns softwares são concebidos com uma organização em módulos e permitem que o usuário programe em uma linguagem própria do software e armazene dados e programas caso haja necessidade de reutiliza-los, garantido assim um ganho de tempo e eficiência.

Com os sitemas de software algébricos, o usuário ganhou em tempo e na precisão dos dados, é claro que saber hoje a previsão de tempo de ontem não faz sentido, a não ser do ponto de vista acadêmico, ou seja, para validar o modelo em questão. Vale resaltar que o avanço tecnológico dos computadores tem permitido o uso de modelos cada vez mais sofisticados, o que permite simular uma gama maior de fenômenos.

6.2 Sugestões para trabalhos futuros

- Utilização de orientação a objetos para:
  - Implementação das demais operações do módulo básico e operações com racionais, citadas no Capítulo 3, usando a estrutura de dados propostas e os algoritmos estudados no capítulo 4 deste trabalho;
  - Implementação do algoritmo de Risch.
  - Implementação das quatro operações básicas para polinômios de uma variável em GF (Galois Fields - Campo de Galois), utilizando os algoritmos e as estrutura de dados estudadas no capítulo 4.
- Elaboração de um tutorial sobre a utilização da computação algébrica como uma ferramenta para modelagem matemática.
- Eleboração de um tutorial sobre o papel da computação algébrica na educação matemática.

Apêndice A

Conceitos de Matemática

A.1 Estruturas Algébricas

Nesta seção serão revistos alguns conceitos de álgebra abstrata, especialmente de grupos e corpos. Serão citados alguns resultados sem provas. Maiores detalhes se encontram em [60, 61, 62].

Álgebra abstrata é um sistema formado por um conjunto S de elementos e um certo número de operações e relações sobre S [60].

Definição A.1.1 *[32]: Um sistema algébrico (álgebra, Ω-álgebra) é um par $[A;\Omega]$ em que*

(1) A é um conjunto;

(2) Ω é uma coleção de operações definidas em A: cada operação é executada por uma função que mapeia $A^n \to A$, levando operandos $a_1, \ldots, a_n \in A$ em $\omega(a_1, \ldots, a_n) \in A$, em que o índice n de ω é um inteiro não negativo que depende de ω, isto é, $n = n(\omega)$. □

Definição A.1.2 *[32]: Um grupóide $(G, \cdot)$ é um sistema algébrico com uma operação binária $\cdot: (x, y) \mapsto x \cdot y$ ($x \cdot y$ também é denotado por xy)*

Um grupóide é multiplicativo ou aditivo se a operação binária for multiplicação ou adição, respectivamente. □

Definição A.1.3 *[32]: Um semigrupo $\langle S, \cdot \rangle$ é um grupóide que satisfaz a lei associativa: $x(yz) = (xy)z$.*□

Exemplos: *Considere os conjuntos definidos como se segue, munidos das operações usuais de adição e multiplicação.*

(a) $\langle Z_+^*; + \rangle$; (b) $\langle Z_+^*; \cdot \rangle$; *em que Z_+^* é o conjunto dos inteiros positivos sem o zero*

(c) $\langle R; + \rangle$: (d) $\langle R; +; \cdot \rangle$: *em que R é o conjunto dos reais.*

(e) $\langle \{0, 1\}; \cdot \rangle$.□

Grupos

Definição A.1.4 *[60]: Um grupo é um conjunto não vazio de elementos S com uma operação binária definida sobre S, aqui denotada por $\circ$, que satisfaz os axiomas seguintes:*

(1) a operação $\circ$ é associativa: para cada tripla (a, b, c) de elementos de S, $(a \circ b) \circ c = a \circ (b \circ c)$;

(2) S possui um elemento identidade à direita na operação $(\circ)$ existe um elemento $\epsilon \in S$, tal que, para cada $a \in S$, $a \circ \epsilon = a$;

(3) para o elemento identidade à direita ϵ, existe para cada $a \in S$ pelo menos um elemento inverso à direita a^{-1}, tal que $a \circ a^{-1} = \epsilon$.□

Se a operação $\circ$ for comutativa, isto é , para cada dupla (a, b) de elementos de S, $a \circ b = b \circ a$, então S é um grupo abeliano.

Um subconjunto não vazio S de um grupo $\langle G, \circ \rangle$ é chamado um subgrupo de G se S for fechado sob $\circ$ e se $\langle S, \circ \rangle$ for um grupo.

Um subgrupo S de G é normal em G se $g \circ S = S \circ g$ para todo $g \in G$.

Anéis

Definição A.1.5 *[60]: Um anel $\langle A, +, \cdot \rangle$ é um conjunto não vazio de elementos A com duas operações binárias definidas sobre A, aqui denotada por $(+)$ e $(\cdot)$, tais que:*

(1) $\langle A, + \rangle$ é um grupo abeliano.

(2) $(\cdot)$ é associativa, isto é, se $\langle A, \cdot \rangle$ for um semigrupo.

(3) Valem as leis distributivas: para todas as triplas (a, b, c) de elementos de A, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ e $(b + c) \cdot a = (b \cdot a) + (c \cdot a)$.□

Seguir-se-á a convenção de usar 0 para indicar a identidade (elemento neutro) de $\langle A, + \rangle$: $(+)$ é adição, $(\cdot)$ é multiplicação.

Um anel é chamado *anel com identidade* se o semigrupo $\langle A, \cdot \rangle$ tiver uma identidade, isto é, se existir um elemento e tal que, $a \cdot e = e \cdot a = a$ para todo $a \in A$. Usar-se-á 1 para indicar, caso exista, a identidade de $\langle A, \cdot \rangle$.

Um anel é *comutativo* se o semigrupo $\langle A, \cdot \rangle$ for comutativo.

Um anel é chamado *anel de integridade* se for um anel comutativo com identidade no qual se $a \cdot b = 0$, então $a = 0$ ou $b = 0$.

Um anel é um *anel de divisão* se os elementos não nulos de A formam um grupo sob $(\cdot)$

Um anel é chamado *corpo* se os elementos não nulos de A formam um grupo comutativo sob $(\cdot)$.

Um subconjunto não vazio S de um corpo K é um *subcorpo* de K, se S com as operações de K formam um corpo.

Um subconjunto S de um anel $\langle A, +, \cdot \rangle$ é chamado um *subanel* de A se S for fechado sob $(+)$ e $(\cdot)$ e se forma um anel sob essas operações.

É comum o uso da notação, $\langle A, +, \cdot, 0, 1 \rangle$, para anel comutativo, onde 0 e 1 são a identidades da operações $+, \cdot$ respectivamente.

Definição A.1.6 *[60]: Um subcorpo não-vazio R de um anel $\langle A, +, \cdot \rangle$ é chamado um ideal se R for um subanel e para todo $r \in R$ e $a \in A$, $ar \in R$ e $ra \in R$.*□

Exemplo: Sejam A um anel comutativo, $a \in A$ e $\langle a \rangle = \{ra + na : r \in A \text{ e } n \in Z\}$. É fácil verificar que $\langle a \rangle$ é um ideal de A; é de fato o menor ideal contendo A. $\langle a \rangle$ é chamado *ideal principal.*

Definição A.1.7 *[62]: Seja K um corpo. P é chamado subcorpo primo de K se P for um subcorpo de todos os subcorpos de K.*□

Definição A.1.8 *[62]: A característica de um corpo K é um inteiro associado a K da maneira seguinte.*

1. Se o corpo primo de K for $\frac{Z}{\langle p \rangle}$, a característica de K é p.

2. Se o corpo primo de K for Q, a característica de K é 0, onde Z é o conjunto dos inteiros e Q é o conjunto dos racionais.□

Freqüentemente denota-se por $\text{car}(K)$ a característica de K. Se $\text{car}(K) = p \neq 0$, então $px = 0$, para todo $x \in K$. Se $\text{car}(K)=0$, então para quaisquer $a, b \in K$ com $a \neq 0$, $ab = 0$ implica $b = 0$.

Definição A.1.9 *[60]: Um Domínio Euclidiano $\langle D, +, \cdot, \nu \rangle$ é um Domínio de Integridade $\langle D, +, \cdot \rangle$ e uma função ν dos elementos não nulos de D no conjunto dos inteiros não negativos, chamada função euclidiana, tal que:*

(1) Para todos os pares a, b de elementos de D para os quais $b \neq 0$ existem q e r em D, tais que:

$$a = bq + r \text{ e } 0 = r \text{ ou } \nu(b) < \nu(a)$$

(2) Para todos os pares a e b de elementos de D para os quais $a \neq 0$ e $b \neq 0$

$$\nu(a) \leq \nu(ab)$$ □

Exemplo: *Os inteiros, com as operações de adição e multiplicação usual formam um domínio Euclidiano onde $\nu(a) = |a|$.*

Se K for um corpo, então $K[x]$ é um domínio euclidiano no qual o grau de um polinômio é uma função euclidiana ν.□

Definição A.1.10 *[61]: Uma função φ: $\langle A_1, +, \cdot \rangle \rightarrow \langle A_2, \oplus, \odot \rangle$, é chamada homomorfismo se para todo $a, b \in A$:*

(1) $\varphi(a + b) = \varphi(a) \oplus \varphi(b)$;

(2) $\varphi(a \cdot b) = \varphi(a) \odot \varphi(b)$.□

Se φ for bijetora, então φ é um isomorfismo e denota-se por $A_1 \cong A_2$. Se $A_1 \cong A_2$, então, A_1 e A_2 se comportam da mesma forma em relação às suas operações.

Extensão de Corpos

Teorema A.1.1 *[60]: Seja K um corpo. Se $p(x)$ for um polinômio não escalar de $K[x]$, existe um corpo $F \supseteq K$ no qual $p(x)$ tem uma raiz.*□

Definição A.1.11 *[60]: Seja $a \in F \supseteq K$. a é transcendente sobre K se $K(a) \cong K(x)$ e a é algébrico sobre K de grau n se a for raiz de um polinômio irredutível $p(x) \in K[x]$ de grau n.*□

Definição A.1.12 *[60]: $F \supseteq K$ é uma extensão algébrica de K se cada elemento de F for algébrico sobre K. Se F não for extensão algébrica então F é uma extensão transcendente de K.*□

Note que se F for uma extensão transcendente de K então F contém pelo menos um elemento que é transcendente sobre K. Aqui F pode ter ou não outros elementos além dos de K que são algébricos sobre K.

Exemplo: *Cada elemento de K é algébrico sobre K, porque se $a \in K$, então a é uma raiz de $x - a$. A maioria dos números reais é transcendental sobre os racionais, entre estes estão π e ϵ.*□

Definição A.1.13 *[60]: Uma extensão $F \supseteq K$ é uma extensão finita de K de grau n se F for um espaço vetorial de dimensão finita n sobre K. Caso contrário F é uma extensão infinita de K. Escreve-se $[F : K]$ para designar o grau de F sobre K.*□

Uma conseqüência da definição implica que se F for uma extensão finita de grau n, então existe um conjunto $\{a_1, \ldots, a_n\} \subseteq F$ tal que $F = K(a_1, \ldots, a_n)$.

Se D for uma extensão finita de F, e F uma extensão finita de K, é conveniente usar um diagrama para ilustrar as extensões de corpos, D fica no topo, como pode ser observado na figura A.1. Assim é possível usar a terminologia sem nenhuma definição precisa de "Torre de Corpos".

Definição A.1.14 *[60]: Um corpo F é algebricamente fechado se F não tiver extensões algébricas próprias.*□

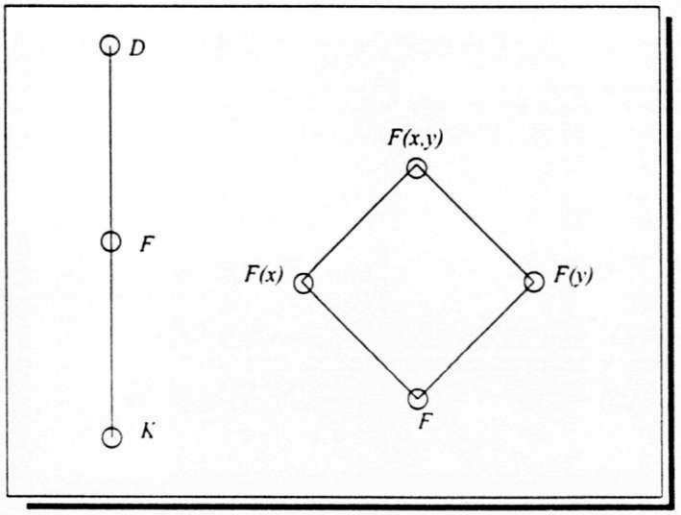

Figura A.1: Estrutura de torre para extensões de corpos

Exemplo: *O corpo dos números complexos é algebricamente fechado. É uma conseqüência do Teorema 1.1 e do Teorema Fundamental da Álgebra, que assegura que cada polinômio com coeficientes complexos tem uma raiz complexa, que por sua vez implica que somente polinômios lineares são irredutíveis.*□

Teorema A.1.2 *[60]: Seja F uma extensão algébrica de K. As condições seguintes são equivalentes.*

1. *F é algebricamente fechado.*
2. *Cada polinômio irredutível de $F[x]$ é linear.* □

A.2 A Transformada Discreta de Fourier e sua Inversa

Serão apresentados alguns conceitos sobre a transformada discreta de Fourier necessários ao entendimento do algoritmo FFT.

Os resultados são apenas citados, sem apresentar nehuma prova formal. Sugestões bibliográficas são fornecidas.

A transformada de Fourier é usualmente definida sobre o conjunto dos números complexos. Aqui ela será definida sobre um anel comutativo $\langle R, +, \cdot, 0, 1\rangle$. Conforme definições 4.3.1 e 4.3.4, neste Apêndice são apresentados alguns resultados e definições adicionais.

Teorema A.2.1 *[29]: Seja $N = 2n$ e ω a N-ésima raiz primitiva da unidade, então $-\omega^j = \omega^{j+n}$.* □

Teorema A.2.2 *[29]: Seja $N = 2n$ e ω a N-ésima raiz primitiva da unidade, então $-\omega^2$ é a n-ésima raiz primitiva da unidade.*□

Uma das aplicações da transformada de Fourier pode ser encontrada no cálculo da convolução de dois vetores, conceitualmente pela definiccão 4.3.4.

Teorema A.2.3 *[29]: Sejam $a = [a_0, a_1, \ldots, a_{n-1}, 0, \ldots, 0]^T$ e $b = [b_0, b_1, \ldots, b_{n-1}, 0, \ldots, 0]^T$, vetores colunas de $2n$ componentes. Seja $F(a) = [a'_0, a'_1, \ldots, a'_{2n-1}]^T$ e $F(b) = [b'_0, b'_1, \ldots, b'_{2n-1}]^T$ suas transformadas de Fourier. Então $a \otimes b = F^{-1}(F(a) \cdot F(b))$. $F(a) \cdot F(b)$ é a operação chamada de produto das transformadas de Fourier, que resulta no vetor $v = [v_0, v_1, \ldots, v_{n-1}, \ldots, v_{2n-1}]^T$, em que cada $v_i = a'_i b'_i$* □

A convolução de dois vetores de n componentes é um vetor de $2n$ componentes. Por isso no Teorema A.2.3, é necessário completar com n zeros os vetores a e b. Para evitar isso, pode-se usar a convolução "camuflada".

Definição A.2.1 *[29]: Sejam $a = [a_0, a_1, \ldots, a_{n-1}]^T$ e $b = [b_0, b_1, \ldots, b_{n-1}]^T$ dois vetores de n componentes. A convolução positiva de a e b é o vetor $c = [c_0, c_1, \ldots, c_{n-1}]^T$, em que:*

$$c_i = \sum_{j=0}^{i} a_j b_{j-i} + \sum_{j=i+1}^{n-1} a_j b_{n+i-j}.$$ □

Definição A.2.2 *[29]: A convolução negativa de a e b é o vetor $d = [d_0, d_1, \ldots, d_{n-1}]^T$, em que:*

$$d_i = \sum_{j=0}^{i} a_j b_{j-i} - \sum_{j=i+1}^{n-1} a_j b_{n+i-j}.$$ □

Teorema A.2.4 *[29]: Sejam* $a = [a_0, a_1, ..., a_{n-1}]^T$ *e* $b = [b_0, b_1, ..., b_{n-1}]^T$ *dois vetores de n componentes. Seja* ω *a n-ésima raiz da unidade e seja* $(\psi)^2 = \omega$. *Supondo que n tem inverso multiplicativo,*

1. *A convolução positiva de a e b é dada por* $F^{-1}(F(a).F(b))$.

2. *Seja* $d = [d_0, d_1, ..., d_{n-1}]^T$ *a convolução negativa de a e b. Sejam* $\hat{a}, \hat{b}$ *e* $\hat{d}$ *respectivamente iguais a* $[a_0, \psi a_1, ..., \psi^{n-1} a_{n-1}]^T$, $[b_0, \psi b_1, ..., \psi^{n-1} b_{n-1}]^T$ *e* $[d_0, \psi d_1, ..., \psi^{n-1} d_{n-1}]^T$. *Então* $\hat{d} = F^{-1}(F(\hat{a}) \cdot F(\hat{b}))$, *com* $\psi^n = -1$. □

É possível avaliar dois polinômios de grau n nas n-ésimas raízes da unidade e multiplicar os pares de valores das raízes correspondentes. Isso fornece n valores com os quais se obtém um único polinômio de grau $2n$. O vetor dos coeficientes do polinômio obtido é exatamente a convolução positiva dos vetores dos coeficientes dos dois polinômios originais [63].

A.3 Congruência Módulo m

A propriedade da divisão de inteiros garante que para $a, m \in Z$ (Conjunto dos Inteiros), $m \neq 0$, existe um único quociente q e um resto r tal que

$$a = mq + r \qquad (0 \leq r < m).$$

Se $r_m(a) = 0$ (o resto da divisão de a por m), então m divide a e denota-se por $m \mid a$, ou seja,

$$m \mid a \iff a = km \text{ para algum } k \in Z.$$

Definição A.3.1 : *Sejam a e b dois inteiros. a é congruente a b módulo m se* $m \mid (a - b) \iff a = km + b$ *para algum* $k \in Z$, *ou seja,* $r_m(a) = b$, *ou ainda,* $r_m(a - b) = 0$. *Representa-se por* $a \equiv b (\text{mod } m)$.□

Verifica-se que a congruência modulo m é uma relação de equivalência, ou seja, valem as seguintes propriedades:

a) $a \equiv a (\text{mod } m)$, a relação é reflexiva:

b) se $a \equiv b(\text{mod } m)$ então $b \equiv a(mod\ m)$. a relação é simétrica;

c) se $a \equiv b(\text{mod } m)$ e $b \equiv c(\text{mod } m)$, então $a \equiv c(\text{mod } m)$. a relação é transitiva.

As congruências gozam, ainda, de duas outras propriedades:

d) se $a \equiv b(\text{mod } m)$ e se z é um inteiro, $az \equiv bz(\text{mod } m)$; e

e) se $a \equiv b(\text{mod } m)$ e se $c \equiv d(\text{mod } m)$, então $ac \equiv bd(\text{mod } m)$ e ainda mais $a + c \equiv b + d(\text{mod } m)$.

Sabendo que a congruência módulo m é uma relação de equivalência então faz sentido definir:

Classe de equivalência de a como $\overline{a} = \{x \in Z \text{ tais que } a \equiv x(\text{mod } m)\}$

Conjunto quociente, como o conjunto de todas as classes de equivalência de Z por m, $Z/m = Z_m = \{\overline{0}, \overline{1}, \overline{2}, ..., \overline{m-1}\}$

Essa relação divide o conjunto dos inteiros em subconjuntos cujos elementos estão relacionados pelo resto que deixam ao serem divididos por m. Por exemplo, definem-se os inteiros *pares* como sendo aqueles congruentes a 0 mod 2. Assim, n é par se e somente se, existe um inteiro m tal que $n = 2m$. Os inteiros *ímpares* são todos os inteiros que não são pares. Pode-se mostrar que um inteiro ímpar n pode ser escrito na forma $2m + 1$, para algum inteiro m. O conjunto quociente para $m = 2$ é $Z_2 = \{\overline{0}, \overline{1}\}$[64].

A.4 Análise Complexa

Nesta seção serão vistos alguns conceitos de análise complexa úteis para a compreensão do capítulo 5. Para maiores detalhes consultar [65].

Quando se trabalha sobre o conjunto dos números complexos é possível considerar quatro tipos de funções: funções reais de variáveis reais, funções reais de variáveis complexas, funções complexas de variáveis reais e funções complexas de variáveis complexas. As letras z e w são usadas para denotar variáveis complexas e para denotar uma função complexa usa-se $w = f(z)$.

A classe de funções analíticas é formada por funções complexas de variáveis complexas diferenciáveis em todo o seu domínio. O termo *função holomórfica* é usado com significado idêntico.

A soma e o produto de duas funções analíticas são também funções analíticas. O mesmo ocorre com o quociente, $f(z)/g(z)$, desde que $g(z)$ não se anule no seu domínio. Nos casos gerais é necessário excluir os pontos em que $g(z) = 0$.

Considere o polinômio,

$$P(z) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

com $a_n \neq 0$. Então $P(z)$ tem grau n. Para $n > 0$, a equação $P(z) = 0$, tem pelo menos uma raiz. Se $P(\alpha_i) = 0$, em que α_i é um elemento do domínio de $P(z)$, é possível escrever $P(z) = (z - \alpha_i)P_i(z)$, em que $P_i(z)$ é um polinômio de grau $n - 1$. A repetição desse processo, completa a fatoração

$$P(z) = a_n(z - \alpha_1)(z - \alpha_2)...(z - \alpha_n)$$

onde $\alpha_1, \alpha_2, ..., \alpha_n$ não são necessariamente distintos.

Se α_i ocorre h vezes, então α_i é uma raiz de $P(z)$ de ordem ou multiplicidade h. Então é possível escrever $P(z) = (z - \alpha_i)^h P_{ih}(z)$, onde $P_{ih}(z) \neq 0$

Considere um função racional,

$$R(z) = \frac{P(z)}{Q(z)},$$

obtida pelo quociente de dois polinômios. Assume-se que $P(z)$ e $Q(z)$ não tenham fatores em comum. $R(z)$ assume valor ∞ nas raízes de $Q(z)$. As raízes de $Q(z)$ são chamadas *polos* de $R(z)$, e a ordem do polo é por definição igual a ordem da raiz correspondente de $Q(z)$.

Definição A.4.1 *[65]: Uma função complexa $f(z)$ é analítica em uma região Ω se ela for definida e tiver derivada de primeira ordem, em cada ponto de Ω. Umafunção é analítica em cada ponto arbitrário de A se ela for analítica em alguma região A.*□

Considere a função $f(z)$ analítica na vizinhança de a, exceto no próprio a. Em outras palavras, $f(z)$ é analítica na região $0 < |z - a| < \delta$. O ponto a é chamado *singularidade isolada* de $f(z)$. Desde de que seja possível calcular $f(a)$, $f(z)$ se torna analítica no disco $|z - a| < \delta$.

Se $\lim_{z \to a} f(z) = \infty$, o ponto é um *polo* de $f(z)$, e escreve-se $f(a) = \infty$. Existe um $\delta' \leq \delta$ tal que $f(z) \neq 0$ para $0 < |z - a| < \delta'$. Nessa região a função $g(z) = \frac{1}{f(z)}$

é definida e analítica. Mas a singularidade de $g(z)$ em a é removível, e $g(z)$ tem uma extensão com $g(z) = 0$. Desde que $g(z)$ não se anule e a raiz em a seja de ordem finita, é possível escrever $g(z) = (z-a)^h g_h(z)$ com $g_h(a) \neq 0$. O número h é a ordem do polo e $f(z)$ tem a representação $f(z) = (z-a)^{-h} f_h(z)$ em que $f_h(z) = \frac{1}{g_h(z)}$ é analítica e não nula em uma vizinhança de a. Observe que a idéia de polo é a mesma usada para funções racionais.

Definição A.4.2 *[65]: Uma função $f(z)$ que é analítica em uma região Ω, exeto nos seus polos, é meromórfica em Ω. Mais precisamente, $\forall a \in \Omega$, deve existir uma vizinhança $|z-a| < \delta$ contida em Ω, tal que toda $f(z)$ é anlítica em toda a vizinhança, ou seja, $f(z)$ é analítica para $0 < |z-a| < \delta$, e a, uma singularidade isolada, é um polo. Observe que os polos das funções meromórficas são isolados pela própria construção da função.*□

O quociente $\frac{f(z)}{g(z)}$ de duas funções analíticas em Ω é uma função meromórfica em Ω, desde de que $g(z) \neq 0$. Os possíveis polos são as raízes de $g(z)$, mas as raízes comuns de $g(z)$ e $f(z)$ podem ser singularidades removíveis. A soma, o produto e o quociente de duas funções meromóficas são também meromórficas. O caso do denominador que se aproxima de zero pode ser excluído, a menos que considere ∞ como uma função meromórfica.

Definição A.4.3 *: Elemento de função ou célula é o par (f, Ω), em que $f(z)$ é função analítica e Ω, o domínio de f.* □

Dois pares (f_1, Ω_1) e (f_2, Ω_2) constituem uma continuação analítica direta uma da outra se $\Omega_1 \cap \Omega_2$ for não vazia e $f_1(z) = f_2(z)$ em $\Omega_1 \cap \Omega_2$. É possível considerar cadeias de elementos de funções (f_1, Ω_1), (f_2, Ω_2), ..., (f_n, Ω_n) tal que (f_k, Ω_k) seja uma continuação analítica direta de (f_{k-1}, Ω_{k-1}): os elementos da cadeia são continuação analítica direta um do outro.

Definição A.4.4 *[65]: Uma função analítica global é um conjunto não vazio **f** de elementos de funções (f, Ω), tais que quaisquer dois elementos em **f** são continuação analítica um do outro formando uma cadeia que une os elementos de **f**. Uma função analítica completa é uma função analítica global que contém todas as continuações de todos os elementos de **f**.* □

A equação da forma $P(z, w) = 0$, em que P é um polinômio em duas variáveis, tem para cada z um número finito de soluções $w_1(z), w_2(z), ..., w_n(z)$.

Definição A.4.5 *[65]: Uma função analítica completa* $\mathbf{f}$ *é uma função algébrica se todos os seus elementos de funções* (f, Ω), *satisfizerem a relação* $P(f(z), z) = 0$ *em* Ω, *em que* $P(w, z)$ *é um polinômio não nulo.*□

Referências

[1] Hattori, M. T. (1992). "*Solução de Problemas de Matemática Usando o Computador*". Relatório Técnico. Departamento de Sistemas e Computação. Universidade Federal da Paraíba, Campina Grande.

[2] Rice, J. R., (1969). "Announcement and Call for Papers, Mathematical Software". *SIGNUM Newslatter*, (4):7.

[3] Hamming, R. W. (1962). "*Numerical Methods for Scientistis and Engineers*". McGraw-Hill, New York.

[4] Morrison, P. and Morrison, E. (editores) (1961) . "*Charles Babbage and his Calculating Engines*". Dover, New York.

[5] Kahrimanian, H. G. (1953). "*Analytical Differentiation by a Digital Computer*". MA Thesis, Universidade de Temple, Philadelphia.

[6] Nolan J. (1953). "*Analytical Differentiation on a Digital Computer*". MA Thesis, MIT, Cambridge, Massachusselts.

[7] Knuth, D. E. (1969). "*The Art of Computer Programming*", volume 1- *Fundamental Algorithms*. Addison-Wesley Publishing, Reading, Massachusetss.

[8] Davenport, J. H.(1985). "Mathematics for Problem Solving Enviroments ". *in Problem Solving Enviroments for Scientific Computing: Symbolic Computation*.Elsevier Science Publishing Company, INC. B.Ford and F.Chatelin, pages 101–115, June.

[9] Rall, L. B. (1981). "Automatic Differentiation - Techinques and Examples". *Springer Lecture Notes in Computer Science, Springer Verlag*, 120.

[10] Geddes K. O. (1980). "Symbolic Mathematical Computation: A Brief Outline of Issues". Dept. of Computer Science, University of Waterloo. September.

[11] Pavelle, R. and Rothsthein, M. and Fitch, J.(1981). "Computer Algebra". *Scientific American*, 245:136-152, December.

[12] Gerd, V. P., Tarason, V. O. e Shirkov D. V. (1980). "Analytic Calculations on Digital Computers for Aplications in Physics and Mathematics". *Sov. Phys. Usp.*, 23(1):59-77, January.

[13] Moses, J. (1971). "Algebraic Simplification: A Guide for Perplexed". *Communications of the ACM*, 14(8):527-537, August.

[14] Barton, D., e Fitch, P. J. (1972). "Preview of Algebraic Manipulative Programs and their Aplications". *The Computer Journal*, 15(4):362 - 378, May.

[15] Tobey, R. (1966). "Experience with FORMAC Algorithm Design". *Communications of the ACM*, 9(8):589-587, August.

[16] Hearn, A. C. (1973). "*REDUCE - User's Manual*". Universidade de Utah.

[17] Caviness, B. F. (1970). "On canonical Forms and Simplification". *Jounal of the Association for Computing Machiner*, 17(2):385-396, April.

[18] Collins, G. (1971a). "The SAC-I system: An Introduction and Survey". *SYMSAM II*, pages 144-152.

[19] Brown, W. S. et al., (1964). "The ALPAK systems for Non-numerical Algebra on a Digital Computer - II". *Bell Sys. Tech. Journal*, 43(2):785-804, March.

[20] Hearn, A. . "A REDUCE 2: A System and Language for Algebraic Manipulation". *Proc. 2nd Symp. on Symbolic and Algebraic Manipulation, ACM Headquarters*, pages 128-135, New York.

[21] Hall, A. D. (1971). "The ALTRAN System for Rational Function Manipulation". *Communications of the ACM*, 14(8):517-521, August.

[22] Tobey, R. et al (1965). "Automatic Simplification in FORMAC". *Proc. AFIPS-FJCC, Spartan Books*, 27(1):37-57, New York.

[23] Fenichel, R. (1966). "*An on-line System for Algebraic Manipulation*". Dissertação de Ph. D., Universidade de Harvard, Cambridge, Massachutts.

[24] Perlis, A et al. (1966). "A Definition of Formula Algol ". *Computer Center, Carnegie-Mellon University*, March, Pittsburgh.

[25] Griesmer, J. H e Jenks, R. D. "SCRATCHPAD/1: An interactivas facility for Symbolic Mathematics". *SYMSAM II*, pages 42-48.

[26] Kowalik, J. S. (1986). "*Coupuling Symbolic and Numerical Computing in Expert Systems*". The Netherlands, North-Holland, Amsterdam.

[27] Kowalik, J. S. (1988). "*Coupuling Symbolic and Numerical Computing in Expert Systems*". The Netherlands, North-Holland, Amsterdam.

[28] Goldberg, D. (1991). "What Every Computer Scientist Should know about Floating-point Arithmetic". *ACM Computing Surveys*, 23(1):963-972, March.

[29] Aho, A. V. et al. (1974). "*The Design and Analysis of Computer Algorithms*". Addison-Wesley Publishing, Reading, Massachusetts.

[30] Knuth, D. E. (1969). "*The Art of Computer Programming*", volume 2 - *Seminumerical Algorithms*. Addison-Wesley Publishing, Reading, Massachusetss.

[31] Kenneth, H. R. (1988). "*Elementary Number Theory and its Aplications*". Addison-Wesley Publishing, Reading, Massachusetts.

[32] Lipson, J.D. (1981). "*Elements of Algebra and Algebraic Computing*". Addison-Wesley Publishing, Reading, Massachusetts.

[33] Horowitz, Ellis e Sahni, Sartaj (1978). " *Fundamentals of Computer Algoritms*". Computer Science Press, Inc, Potamac, Maryland.

[34] Baase, Sara (1978). " *Computer Algorithms - Introdution to Design and Analysis*". Addison-Wesley Publishing, Reading, Massachutsets.

[35] Collins, E. G. (1966). "PM, A System for Polynomial Manipulation". *Communications of the ACM*, 9(8):578-589, August.

[36] Hyde, J. P.(1964). " The ALPAK System for Non-numerical Algebra on a Digital Computer - III: Systems of Linear Equations and a Class of Side Relations". *Bell Sys. Tech. Journal*, 14:1547-1562, July.

[37] Brown, W. S. (1963). "The ALPAK system for non-numerical algebra on a digital computer - I: Polynomials in several variables and truncated power series with polynomial coefficientes". *Bell Sys. Tech. Journal*, 42:2081-2119, September.

[38] Hyde, J. P. and Tague, B. A. (1964). "The ALPAK System for Non-numerical Algebra on a Digital Computer - II: Rational Functions of Several Variables and Truncated Power Series with Rational Function Coefficients". *Bell Sys. Tech. Journal*, 43:785-804, March.

[39] Collins, E. G. (1967). "Subresultants and Reduced Polynomial Remainder Sequences". *Journal of the Association for Computing Machinery*, 14:128-421, May.

[40] Borwein, J. M. e Borwein, P. B.(1984). " The Aritmetic-Geometric and Fast Computation of Elementary Functions". *SIAM REVIEW*, 26(3):1-19, July.

[41] Tobey, R. (1967). "*Algorithms for Antidiferentiation of Rational Functions*". Ph.D. diss, Harvard University, Cambridge, Massachusetts.

[42] Slagle, J. A. (1961). "*A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus*". Ph.D. diss, MIT, May.

[43] Moses, J. (1967). " Simbolic Integration MAC-TR-47". *Proj MAC MIT*, December.

[44] Manove, M., Bloom, S. and Engelman, C. (1968). "Rational Functions in MATHLAB". *PROC. IFIP Conf. on Symbolic Manipulation Languages*, Pisa, Italy.

[45] Horowitz, E. (1971). "Algorithms for Partial Fraction Integration". *Proc. Second Symposium on Simbolic and Algebraic Manipulation-SYMSAM II. ACM Head quarters*, 23:441-457, New York, March.

[46] Richardson, D. (1968). "Some Unsolvable Problems Involving Elementary Functions of a Real Variable". *J. Symbolic Logic*, 33:511-520.

[47] Risch, R. H. (1968). "On the Integration of Elementary Funtions which are Built up Using Algebric Operations". *Systems Development Corporation*, (Rep. SP-2801-002), June, Santa Mônica, California.

[48] Rosenlight, M. (1972). "Integration in Finite Terms". *Amer. Math. Monthy*, (79):963-972, November.

[49] Risch. R. H.(1969). "The problem of Integration in Finite Terms". *Transactions American Mathematical Society*, (139):162-189, May.

[50] Risch. R. H.(1970). "The Solution of the Problem of Integration in Finite Terms". *Bulletion of AMS*. 139:605-608, May.

[51] Rosenlight, M. (1968). "Liouville's Theorem on Functions with Elementary Integrals ". *Pacific Journal of Mathematics*. 24:153-161.

[52] Davenport. J. H.(1981). "On the Integration of Algebraic Functions". *Springer Lecture Notes in Computer Science*. 102.

[53] Davenport. J. H.(1982). "On Parallel Rich Algoritm (I)". *Springer Lecture Notes in Computer Science*. 143:144-157.

[54] Trager. B. M.(1979). "Integration of Simple Radical Extension". *Springer Lecture Notes in Computer Science*. 79:408-414.

[55] Davenport. J. H.(1984). "Intégration Algorithmique des Fonction Élémentairement Transcendantes sur une Coubre Algébrique". *Annales de Institut Fourier*. 34:271-276.

[56] Trager. B. M. (1984). "*Integration of Algebraic Functions*". Ph. D. Thesis. M.I.T. Dept. of EECS. August.

[57] Rayana G. (1987). "*REDUCE - Software for Algebraic Computation*". Spring-Verlag. New York.

[58] Bruce W. C. et al. (1988). "*MAPLE Reference Manual*". Waterloo Maple Publishing. Ontario.

[59] Wolfram S. (1991). "*Mathematica - A System for Doing Mathematics by Computer*". Segunda Edição. Addison-Wesley Publishing. Redwood City. California.

[60] Dean. R. (1974). "*Elementos de Álgebra Abstrata*". Livros Técnicos e Científicos Editora S.A.. Rio de Janeiro.

[61] Fraleigth. J. B. (1968). "*A First Course in Abstract Algebra* ". Addilson-Wesley. Reading. Massachutsetts.

[62] Hernstein . I. N. (1970). " *Tópicos de Algébra*". Editora da Universidade de São Paulo e Polígono. São Paulo.

[63] Goldberg, Richard, R. (1970). "*Fourier Transforms*". Cambridge University Press, Cambridge.

[64] Viswanathan, T. M (1979). "*Introdução a Álgebra e a Aritmética*". Instituto de Matemática Pura e Aplicada. Rio de Janeiro.

[65] Alfhors L. V. (1966). "*Complex Analysis*". McGraw-Hill Koga kusha, Tokyo, Japan.