Computação Algébrica: 'Um Assistente Matemático'

R. P. dos Santos

Resumo

Neste trabalho, apresentamos, de forma simples e informativa, o tema "Computação Algébrica", visando despertar e estimular a comunidade brasileira de Ciência e Tecnologia para o uso desta nova ferramenta.

O objetivo da Computação Algébrica é propiciar a execução, em computador, de cálculos com variáveis indefinidas, de maneira eficiente e exata.

Vários sistemas de computação algébrica vêm sendo desenvolvidos e aplicados em muitos campos de pesquisa. Alguns desses sistemas, suas linguagens de implementação e aplicações são, também, discutidos.

Introdução

Em muitas áreas da atividade humana, fazemos uso de números. Freqüentemente, este uso é intenso e ,por isso, o homem vem procurando desenvolver métodos artificiais para auxiliá-lo. Provavelmente, após o uso dos dedos, o primeiro dispositivo artificial de cálculo foi o ábaco. Sucedendo ao ábaco, Charles Babbage criou, no fim do século 19, a Máquina Analítica, que apresentava muito mais recursos. Depois, teve-se a régua de cálculo e hoje estamos acostumados a conviver com computadores gigantescos e calculadoras de pulso.

Todavia, estamos acostumados também a pensar que essas máquinas podem realizar cálculos apenas com números. Parecemos acreditar que o nível de abstração que a mente humana precisou atingir para conceber o cálculo com quantidades indefinidas – a álgebra – torna-o proibitivo para essas mesmas máquinas.

O fato, porém, é que os computadores **podem** trabalhar com símbolos e não há nenhuma novidade nisso. Estamos habituados a cadastros computadorizados de nomes, de títulos de livros, etc.

Aliás, já em 1844, Lady Lovelace, protetora de Charles Babbage, previu a possibilidade de sua máquina manipular símbolos. No entanto, só em 1953 conseguiu-se construir um sistema que realizasse cálculos algébricos simples: diferenciando-se $\mathtt{sen}\ x^2$ obteve-se $2\ x\ \cos\ x^2$. Note-se que x é uma quantidade indefinida, sendo o cálculo acima válido para qualquer valor de x.

É interessante ressaltar que o primeiro sistema capaz de realizar integrações indefinidas (uma tarefa muito mais difícil que diferenciação, uma vez que não há uma estratégia definida a priori para aquela) foi elaborado por um pesquisador interessado em estudar os processos da simulação da inteligência humana.

Hoje, felizmente, o profissional pode contar com sistemas computacionais algébricos, poderosos auxiliares que lhe oferecem recursos matemáticos equivalentes a uma verdadeira enciclopédia de técnicas de cálculo na ponta dos dedos, aliados a capacidade de tratar expressões realmente enormes, a uma velocidade muito superior a do ser humano e, principalmente, praticamente sem possibilidade de cometer erros. Fazem o papel, no dizer de Joel Moses, de um "assistente matemático".

Um exemplo da potência de cálculo desses sistemas, é a repetição em computador dos cálculos efetuados por Charles Delaunay, no fim do século passado, referentes a posição da Lua em sua órbita como função do tempo. À mão, Delaunay levou 10 anos para efetuar esse cálculo **e mais 10 anos verificando-o**, tendo publicado os resultados em dois volumes.

Hoje, esses cálculos podem sem refeitos em computador **em menos de um minuto**, verificando-se, essencialmente, apenas um erro em todos os dois volumes de resultados (algo realmente notável).

Num exemplo da confiabilidade que se confere hoje em dia a esses sistemas, num trabalho de E. Grahan e D. R. Moore, no qual reduziram as equações de movimento de um fluido a um conjunto de equações diferenciais lineares simples, a observação de que os sistemas **CAMAL** e **FASM** foram utilizados para o cálculo foi deslocada do parágrafo de encerramento para a introdução, para que o leitor utilizasse o resultado com confiança, não perdendo tempo em conferi-lo.

Apesar dos sistemas algébricos terem sido usados em áreas tão distintas do conhecimento humano como Relatividade Geral, Teoria dos Números, Química, Economia e Processamento de Imagens, para citar apenas algumas, somente uma pequena parcela dos usuários potenciais tem desfrutado de seus recursos, desperdiçando tempo e esforço, realizando manualmente cálculos que poderiam ser obtidos rapidamente e com absoluta confiabilidade.

Talvez, um dos motivos para que isso ocorra venha a ser o fato de que as linguagens de computador mais divulgadas (FORTRAN, COBOL, etc.) não são apropriadas para cálculos algébricos.

Os sistemas algébricos mais poderosos, atualmente em uso, são codificados em linguagens como LISP e C, as quais trabalham com listas ou conjuntos de elementos (chamados átomos), ao invés de com números. Porém, o fato do pesquisador não estar familiarizado com essas linguagens não justifica sua não familiaridade com os sistemas algébricos, pois não é, na verdade, necessário conhecê-las para poder usá-los.

Estes sistemas, em geral, propiciam uma comunicação fácil com o usuário, facilidade essa que é maior ainda no caso dos sistemas especializados, nos quais o prejuízo da generalidade é plenamente compensado por recursos específicos e por uma notação muito mais próxima aquela que seu usuário emprega nos cálculos a mão.

No dizer de Inge Frick, da Universidade de Estocolmo, sobre a utilização do sistema **SHEEP**, especializado em Relatividade Geral, "você simplesmente fala com ele corretamente e ele lhe responde em seguida".

De fato, o usuário iniciante precisa conhecer apenas alguns comandos básicos do sistema algébrico e alguns do próprio computador (como ligar o terminal, como acessar o sistema, etc.). Este usuário usa o sistema algébrico como uma espécie de calculadora algébrica, calculando integrais, resolvendo equações diferenciais, etc.

Numa fase posterior, o usuário, para tirar maior proveito do sistema e para construir algoritmos mais eficientes, procura conhecer mais a fundo todos os recursos que o sistema oferece. Este usuário constrói verdadeiros programas, usando o sistema como uma linguagem de programação.

Finalmente, o usuário chega ao ponto de necessitar recursos que o sistema não oferece. Neste ponto, os sistemas flexíveis permitem que o usuário incorpore facilidades novas, codificadas, freqüentemente, na própria linguagem de implementação. Aqui, o usuário pode passar a ser um colaborador do criador do sistema, desenvolvendo-o.

Outro motivo que dificulta a difusão dos sistemas algébricos é, certamente, sua exigência de memória do computador. Enquanto um pequeno microcomputador de 8 bits, com 64 Kbytes de memória, ou mesmo uma calculadora sofisticada podem realizar cálculos numéricos úteis, os mesmos serão capazes de realizar apenas cálculos algébricos triviais.

Para um cálculo realmente sério, será necessário 1 megabyte de memória e um pesquisador provavelmente necessitará 2 megabytes ou mais, o que está muito acima da capacidade de um micro, atingindo o nível das máquinas chamadas supermicros ou superminis. No entanto, um micro PC-compatível, com 640 Kbytes, já poderá realizar cálculos algébricos interessantes e existem versões de sistemas como **muMATH** para eles, estando também sendo feitas versões de **REDUCE** e **SCHOONSHIP**.

Computação Numérica versus Computação Algébrica

Na introdução, apareceu um exemplo de computação algébrica, uma diferenciação. Vejamos, agora, mais alguns exemplos para que fique claro a diferença com relação a computação numérica:

No exemplo 1, expandimos uma expressão que resulta num polinômio de alto grau. O procedimento consistiu em abrir os parênteses, efetuando as exponenciações e multiplicações correspondentes, simplificar os termos de potência semelhante e escrever o resultado em ordem decrescente da potência da variável. Tudo isso foi realizado automaticamente pelo programa algébrico, bastando apenas digitar a expressão de entrada e ele respondendo com a expressão simplificada. Na verdade, ele imprime todos os 71 termos do polinômio. Por edição, deixamos apenas alguns termos típicos. Note, também, que os coeficientes são números arbitrariamente grandes.

```
1: (x-1)**20*(2*x-1)**20*(3*x-1)**30;

21589249972727866982 4*x**70 - ... +
41869228239514176855083224960549 6*x**40 +...+
+ 534866040*x**5 + 19078420*x**4 - 535080*x**3 + 11065*x**2 - 150*x + 1
```

Exemplo 1 Expandindo um polinômio de alto grau

No exemplo 2, apresentamos alguns exemplos de diferenciação simbólica. As funções `dilog` e `erf` são as funções matemáticas dilogaritmo e função erro.

```
2: DF(COS(X),X);

- SIN(X)

3: DF(ATAN(Y),Y);

1/(Y**2+1)

4: DF(EXPINT(Z),Z);

E**Z/Z

5: DF(DILOG(U),U)

- LOG(X)/(X-1)

6: DF(ERF(X),X);

(2*SQRT(PI))/(E**(X**2)*PI)
```

Exemplo 2 Diferenciação

No exemplo 3, temos uma integração simbólica. Note o tempo de CPU que o sistema levou para esse cálculo: menos de meio segundo!

```
7: INT(LOG(X)**5,X);

X*(LOG(X)**5 - 5*LOG(X)**4 + 20*LOG(X)**3 - 60*LOG(X)**2 + 120*LOG(X) - 120)
```

Exemplo 3 Integração

No exemplo 4, demonstramos que as linguagens simbólicas podem também, em geral, trabalhar com números. Esses sistemas, via de regra, são mais lentos, para esse tipo de cálculo, que linguagens essencialmente numéricas como **FORTRAN**, mas permitem uma precisão muito grande. Como exemplo, temos o número `pi` com 50 dígitos de precisão (e poderíamos apresentá-lo com precisão ainda maior!). Em seguida, calculamos coseno (π /6), também com precisão de 50 dígitos. Para verificar a exatidão do cálculo, cujo resultado deveria ser a raiz quadrada de 3 dividida por 2, elevamos o

resultado obtido ao quadrado. Note que, como o resultado é exato, temos apenas dois dígitos, embora a precisão admitida continue sendo de até 50 dígitos.

```
8: PRECISION 50; PI;

3.14159265358979323846264338327950288419716939937 51

9: COS(PI/6);

0.86602540378443864676372317075293618347140262690519

10: WS**2;

0.75
```

Exemplo 4 Precisão absoluta

No exemplo 5, obtemos um número bastante grande, num exemplo de cálculo com inteiros.

```
11: 2**1000;

10715086071862673209484250490622218105614048117055360744375038

83703510511249361224931983788156985812759467291175531468251871

45285692314043598457757469857480393456774824230985421074605062

37114182153046477498358194126673987675591165543946077062914571

19647768654216766042983165262438688372056680693 76
```

Exemplo 5 Cálculo com grandes números

Um ponto importante a se levar em conta em computação algébrica é a ocupação de memória. Inicialmente, tem-se que uma expressão algébrica, por mais simples que seja, ocupa mais memória que um número em ponto flutuante em linguagens convencionais como **FORTRAN**. Além disso, há, também, o problema dos resultados intermediários que ocupam bastante memória (vide a seção sobre LISP).

Outro aspecto relevante dos sistemas algébricos baseados em linguagens como LISP e C é a característica da recursão, isto é, um programa ou função referir-se a si mesmo na sua definição. O exemplo clássico de recursão é a definição do fatorial:

$$0!=1$$

$$n! = n(n-1),\ n > 0$$

Isso permite, freqüentemente, a codificação de programas mais elegantes e, por vezes, só se possui uma definição recursiva para a função desejada.

No entanto, em alguns casos, esse recurso não é conveniente, pois forma-se uma cadeia de resultados pendentes da definição recursiva até se encontrar um resultado que pode ser retornado no caminho inverso, obtendo-se o resultado final. Esquematicamente, teríamos, por exemplo, para o fatorial de 4:

```
4! (cálculo proposto)
4*3! (aplicação recursiva)
4*3*2!
4*3*2*1!
4*3*2*1*0! (encontrado o cálculo definido 0!)
4*3*2*1*1 (retorno da recursão)
4*3*2*1
4*3*2
4*6
24 (fim do cálculo)
```

Até ser encontrado o resultado definido 0!, foi sendo construída a cadeia de resultados pendentes intermediários, consumindo memória. No caso de uma definição não recursiva do fatorial, ao estilo FORTRAN, teremos uma seqüência de multiplicações, usando sempre a mesma quantidade de memória, sem resultados pendentes.

Essa utilização maior de memória pode ser catastrófica. No exemplo 6, apresentamos definições recursiva e iterativa do fatorial e aplicações, verificando-se a sobrecarga de memória no caso recursivo.

A linguagem LISP

A linguagem LISP trabalha com listas de elementos, os quais podem ser outras listas ou elementos simples (chamados átomos). Exemplos de listas são:

(A B C D)

((RENATO PIRES) (CBPF RIO))

Tem-se em LISP funções para manipulação de listas. Por exemplo, a função `CAR` dá, como resultado, o primeiro elemento de uma lista, CDR retorna o restante da lista, retirado o primeiro elemento; estas funções podem ser combinadas como `CADR` (`CAR` do`CDR`), `CDDR` (`CDR` do `CDR`), etc. Tem-se também predicados lógicos como: `EQ` que

devolve o valor `true` (verdade) se os átomos comparados são iguais e `nil` (falso) se diferentes,`ATOM` que devolve `true` se o elemento analisado é um átomo e `nil` caso contrário. Tem-se ainda `IF` que testa uma condição e retorna o primeiro argumento fornecido se a condição for verdadeira e o segundo se falsa e `COND` que testa uma série de condições devolvendo o valor do argumento correspondente a primeira condição que se verificar verdadeira. Note que os comandos de LISP são, por sua vez, também listas.

Como exemplos do funcionamento dessas funções, temos

(CAR (A B C D)) = A

(CDR (A B C D)) = (B C D)

(CADR (A B C D)) = B

(CDDR (A B C D)) = (C D)

(CAR ((RENATO PIRES) (CBPF RIO)) = (RENATO PIRES)

(CDAR (RENATO PIRES) (CBPF RIO)) = (PIRES)

(EQ (A A)) = true

(ATOM (A B C D)) = nil

(IF (ATOM A) A (CAR A)) = A

(COND (c_1 v_1) (c_2 v_2) (c_3 v_3) ...)

Em LISP, expressões algébricas, contrariamente a linguagens como FORTRAN, são escritas em notação prefixada, isto é, o operador vem à frente dos operandos. Por exemplo, temos

notação usual	notação FORTRAN	notação LISP
x^2+x+3	`x*x+3*x+3`	`(+ (+ (*x x) x) 3)`

Esta expressão, por demais simples, ocuparia, no código mais comum dos usados para armazenamento de expressões, no qual se utiliza 1 byte para cada caractere, 6 bytes. Em LISP, seriam necessários no mínimo 18 bytes, considerando-se que cada átomo necessite de 2 bytes para ser armazenado.

Em computação numérica, uma posição de memória com 4 bytes pode armazenar um número inteiro de magnitude máxima 2.147.483.647 ou um número em ponto flutuante de magnitude entre 10^{-78} e 10^{+76}, com 7 dígitos de precisão, o que é suficiente para a maioria dos cálculos numéricos. Por isso, não é incomum um cálculo algébrico necessitar de 2 megabytes ou mais. Além disso, durante um cálculo a ocupação de memória por cálculos intermediários pode ser desastrosa. Por exemplo, seja a expansão de $(x+a)^3$:

$$(x+a)^3 = (x+a)(xx + xa + ax + aa)$$
$$= (xxx + xxa + xax + xaa + axx + axa + aax + aaa)$$

$$(xxx + axx + axx + aax + axx + aax + aax + aaa)$$

$$(xxx + axx + axx + axx + aax + aax + aax + aaa)$$

$$(x^3 + ax^2 + ax^2 + ax^2 + a^2x + a^2x + a^2x + a^3)$$

$$(x^3 + 3ax^2 + 3a^2x + a^3)$$

Temos, nas primeiras duas linhas, a abertura dos parênteses, efetuando as multiplicações correspondentes. Em seguida, alfabetiza-se cada termo, colocando os termos de cada produto em ordem. Efetuam-se os produtos e alfabetizam-se, então, os termos da expressão. Finalmente, simplificam-se os termos semelhantes, obtendo o resultado desejado.

O ponto a se ressaltar aqui é que as listas contendo os resultados intermediários só serão descartadas após a obtenção do resultado final, por um dispositivo chamado coletor de lixo ("*garbage collector*"). Considerando-se a memória necessária para se armazenar cada termo de cada lista, podemos calcular a quantidade de memória utilizada para um cálculo tão simples como este. Imagine, então, a quantidade necessária para um problema real bem mais complexo!

```
deriv(expr1 + expr2) = deriv(expr1) + deriv(expr2)
deriv(expr1*expr2) = deriv(expr1)*expr2 + expr1*deriv(expr2)
```

Vejamos, agora, como é possível fazer-se uma manipulação mais sofisticada de uma expressão algébrica. Consideremos um pequeno programa LISP (vide exemplo 7) que deriva polinômios, implementando as regras mais simples de derivação:

```
simp(átomo) = átomo
simp(lista1 + lista2) = simp(lista1) + simp(lista2)
simp(lista1 * lista2) = simp(lista1) * simp(lista2)
simp(número1 + número2) = número1 + número2
simp(número1 * número2) = número1 * número2
```

Temos, então, usando as funções LISP anteriormente vistas, um pequeno programa LISP que realiza derivações sobre polinômios.

Se aplicado este programa a expressão LISP anteriormente comentada, obteríamos

notação LISP	notação usual
(+ (+ (+ (* 1 x) (* x 1)) 1) 0)	1x + 1x + 1 + 0

Note que o resultado, embora correto, não está simplificado. A simplificação em computação algébrica está, na verdade, longe de ser um problema trivial. Não é fácil caracterizar-se o procedimento de simplificação e, além disso, não há um senso comum

sobre quando uma expressão está na sua forma mais simples. A "forma mais simples" pode variar, segundo o contexto do cálculo e o gosto do usuário.

Certas regras básicas, tais como

$$x + 0 \rightarrow x$$

$$0 * x \rightarrow 0$$

$$1 * x \rightarrow x$$

devem, quase sempre, ser utilizadas pelo sistema. Considerando-se as regras acima, poderíamos conceber o simples procedimento de simplificação apresentado no exemplo 8, que leva em conta, também as seguintes regras:

Todavia, tal procedimento não é completo, uma vez que, se aplicado ao resultado anterior, retornaria `(+ (+ x x) 1)`, o que representaria, na notação usual, `x+x+1`. Da mesma forma, ele não simplificaria `1*1`, por exemplo. Se, no entanto, a última linha do procedimento for alterada para que simplifique os termos do produto, poderia ocorrer uma recursão infinita no caso de um produto de termos já simplificados como`3*x`.

Em geral, deve-se ter certo discernimento na aplicação de uma regra de simplificação, pois é freqüente sua aplicação impensada levar a uma expressão mais complicada que a original.

Hoje em dia, trabalha-se no desenvolvimento de procedimentos heurísticos de simplificação, que avaliem se e onde uma simplificação deve ser feita. De qualquer forma, a pessoa mais indicada para decidir qual a forma simplificada de uma expressão é o próprio usuário. O sistema algébrico deve oferecer vários recursos de simplificação, decidindo a utilização de alguns e permitindo ao usuário a utilização dos que julgar convenientes, ou mesmo impedir a sua aplicação automática.

Integração analítica por computador

Certamente, uma das áreas de desenvolvimento mais interessantes em computação algébrica é a integração. Muitas pessoas, quando ouvem falar que o computador pode realizar integração analítica, imaginam que ele tenha sido programado com uma tabela de integrais e que o sistema simplesmente faca uma espécie de busca em tabelas. Mas o que se tem, é algo bem mais sofisticado e eficiente.

Existem quatro abordagens básicas a uma integral, por parte de um sistema algébrico:

1. **Busca a padrões:** O sistema faz uma inspeção da integral, procurando identificar integrais básicas que possam ser realizadas imediatamente. Esta primeira abordagem é de certa forma, uma pesquisa em tabela.
2. **Métodos heurísticos:** O sistema tenta, aqui, recursos como mudanças de variáveis e integrações por partes, até que se caia em integrais básicas que possam ser tratadas pela abordagem anterior.
3. **Métodos especializados:** Aqui, utilizam-se procedimentos específicos, tais como expansão em frações parciais, no caso de funções racionais, etc...

4. **Método geral:** Esta é a abordagem mais interessante, tanto do ponto de vista computacional como mesmo matemático. Quando se iniciou o desenvolvimento dos sistemas de integração analítica em computador, considerava-se que a integração não fosse um processo algorítmico como a diferenciação. Todavia, num trabalho que se inicia com Laplace, no começo do século passado, passando por Abel e Liouville, foi possível chegar-se ao algoritmo de Risch e Norman que é capaz de decidir se a integral de uma função *f* pertencente a um campo de funções elementares *F* pode ser escrita como

$$\int f\,dx = V_0 + \sum C_i \log V_i$$

onde V_0 e $V_i \in F$ e C_i são constantes, e também determinar V_0, V_i e C_i.

A integral que puder ser resolvida por uma das três primeiras abordagens, o será eficientemente, pois o sistema passa rapidamente por elas, tendo sucesso ou não. Mas quando não tiverem sucesso, não informarão se a integral pode ou não ser escrita na forma de uma combinação de funções elementares do campo delimitado.

Já a última abordagem permite um procedimento de decisão: se a integral não puder ser expressa em termos de funções elementares, o sistema constata essa situação e a reporta.

É interessante notar, porém, que o algoritmo de Risch e Norman, embora seja capaz de obter a expressão da integral, quando possível, exigirá considerável trabalho, exceto nos casos mais simples que podem ser resolvidos mais eficientemente pelas outras abordagens. Sendo, todavia, um processo algorítmico, essa abordagem presta-se muito bem para ser implementada em computador.

Sistemas de Computação Algébrica

Embora a Computação Algébrica tenha dado seus primeiros passos há cerca de 30 anos, só recentemente vem desabrochando e se espalhando, com aplicações nos mais variados campos de pesquisa em Ciência e Tecnologia.

Existem, hoje, vários sistemas para processar cálculos algébricos, escritos em diferentes linguagens de computador. Mais de 40 sistemas são conhecidos com esta finalidade, tendo sido, em sua maioria, criados visando aplicações específicas em problemas de Astronomia, Relatividade Geral e Física das Partículas Elementares.

Além das aplicações, estes sistemas se diferenciam, também, pelo nível da linguagem em que estão escritos. As linguagens de programação podem ser divididas em pelo menos três níveis:

1. **Linguagem de Máquina**. Esta linguagem é constituída de instruções que são diretamente assimiladas pelo computador (máquina). Desta forma, elas são máquina-dependentes, isto é, dependem da marca e do tipo do computador.
2. **Linguagem Montadora (*Assembly*)**. Esta linguagem, embora ainda próxima da linguagem de máquina, é mais inteligível pelo usuário

precisando, no entanto, ser traduzida para aquela, por um programa chamado compilador, de forma que possa ser entendida pelo computador. Note-se que esta linguagem também é máquina-dependente.

3. **Linguagem de Alto-Nível**. Esta é a que mais se assemelha a nossa linguagem coloquial. Na linguagem de alto-nível, os programas podem ser escritos, por exemplo, em português e com notação matemática, devendo, porém, ser traduzidos por um compilador. Em princípio, as linguagens de alto-nível são máquina-independentes e padronizadas internacionalmente. Como devem ser traduzidas por compiladores máquina-dependentes, na prática, são necessárias ligeiras modificações quando se transfere um programa de uma máquina para outra, de fabricante diferente.

Os primeiros sistemas de computação algébrica foram baseados em linguagens montadoras e, posteriormente, nas linguagens de alto-nível FORMAC (uma extensão do FORTRAN para manipulação algébrica) e LISP. Em particular, os sistemas com linguagem de implementação LISP vem recebendo, nos últimos anos, uma atenção muito maior do que os sistemas com base em FORMAC.

Bibliografia

Pavelle, R., Rothstein, M., Fitch, J. *Computer Algebra*, Scientific American **245**(6):102-113, 12/1981