Tutorial sobre Algebra Computacional com Programação

Leandro Rangel[1] e Alexandre Rojas[2]
[1]Aluno do Bacharelado em Matemática- UERJ
[2] Docente Departamento de Informática e Ciencia da Computação- UERJ

Resumo

O presente Tutorial tem por objetivo apresentar uma breve visão relativa aos principais comandos utilizados nos softwares de Computação Algébrica em especial o Mupad, desenvolvido pela Universidade de Paderborn – Alemanha destacando a linguagem de programação contida neste software.
Trata-se de um projeto multidiscipliar integrando competencias da área matemática com a Ciência da Computação.

1. Introdução

Até a década de 1990, a grande maioria dos professores de matemática utilizava como suporte metodológico, na elaboração de suas atividades docentes, basicamente o livro didático. A partir do advento da popularização da informática uma gama maior de possibilidades passou a existir, permitindo que o ensino matemático fosse difundido com maior facilidade.
O processo de mudança no processo de ensino e aprendizagem é caracterizado por resistências, principalmente na inserção de novas tecnologias no processo educacional. Sancho (2001, p. 43) refere-se a essas resistências como "tecnofobia". Porem em diversas universidades existe um movimento para a utilização de softwares educacionais em diversas disciplinas, dentre elas o calculo diferencial e integral.
O uso de ferramentas computacionais no processo de ensino-aprendizagem permite ao estudante abordar problemas complexos. No caso da computação simbólica, sistemas computacionais permitem tornar a Matemática mais experimental, permitindo analisar diferentes situações com visualização gráfica, dando assim oportunidade ao estudante de aprender fazendo.

2. Definições

2.1. Computação Algébrica

A computação algébrica é a área da computação que lida com a manipulação e solução exata de equações. Estas equações podem ser polinomiais, diferenciais ou a diferenças finitas, por exemplo. Trata-se de uma área muito ampla e, portanto, com várias vertentes, as mais conhecidas são: testes de primalidade e algoritmos de fatoração para inteiros e polinômios, manipulação e solução de sistemas de equações polinomiais em várias variáveis, cálculo exato de integrais indefinidas de funções elementares e cálculo da forma fechada de somatórios de números binomiais [1]. Cada uma destas vertentes tem métodos próprios, oriundos de uma área da matemática subjacente, que nos casos mencionados acima, são respectivamente: teoria de números, teoria das equações algébricas, geometria algébrica, cálculo diferencial e combinatória. Entretanto, em quase todas as vertentes, estes métodos se misturam com técnicas de áreas afins e de computação. Assim, por exemplo, o método de bases de Groebner, inventado no contexto da geometria algébrica, foi generalizado para lidar com álgebras de operadores, e através destas aplicado a problemas de identidades combinatórias.

2.2. Sistemas de Computação Algébrica

Um sistema de computação algébrica (ou simbólica) é um software que permite manipular expressões matemáticas de maneira simbólica. Normalmente estes sistemas permitem efetuar cálculos com inteiros de precisão múltipla (isto é, sem que haja um tamanho especifico a priori para o maior inteiro), frações expressas como a razão entre dois inteiros e polinômios com uma ou várias variáveis.
A maioria dos sistemas de computação algébrica atuais pode ser utilizada de maneira interativa. O usuário entra com algumas fórmulas e comandos, e o sistema os avalia. Então devolve uma resposta que pode ser manipulada mais adiante se necessário.
Além de computações simbólicas exatas, os sistemas de computação algébrica podem obter soluções numéricas aproximadas. O usuário pode fixar a precisão no número de dígitos desejados. Os sistemas de computação algébrica modernos possuem linguagens de programação poderosas, alem de ferramentas para visualização e animação de dados matemáticos.

2.3. Computação simbólica

A computação simbólica é um ramo da Ciência da Computação e da Matemática cujos fundamentos teóricos centralizam-se no estudo de não numéricos, isto é, as operações simbólicas que podem tratados por um computador, com ênfase especial em cálculos simbólicos tais como fatoração de polinômios, resolução de equações algébricas e equações diferenciais, operações e cálculo com matrizes, etc.
Os cálculos realizados no tratamento simbólico são exatos, isto é, têm precisão infinita, em contraste ao correspondente tratamento numérico. Uma operação do tipo 1/3+1/3, que numericamente resultaria em 0.666666, no cálculo simbólico teremos como resultado o valor exato, 2/3.
Nas ultimas décadas muitos sistemas de computação simbólica foram desenvolvidos. Os aplicativos mais conhecidos são Axiom, Derive, Macsyma, Maple, Mathematica, Reduce e MuPAD.
MUPAD
A ferramenta de Apoio utilizada nesse trabalho é o MUPAD. (Multi Processing Data Tool), [2] caracterizado como um Sistema de Computação Algébrica desenvolvido na Universidade de Paderborn na Alemanha. O projeto teve início em setembro de 1989, com a tese de mestrado de Karsten Morisse e Oliver Kluge. Andreas Kemper, em sua tese de mestrado, desenvolve a linguagem de programação MUPAD, semelhante em diversos aspectos ao Pascal — linguagem de programação de aprendizado relativamente fácil. Utilizado extensamente em muitas universidades européias e asiáticas como uma ferramenta educacional, na realidade, MUPAD contém uma linguagem de programação completa. A linguagem de programação do MUPAD compartilha as características mais comuns de muitas linguagens de programação. utiliza 'se-então', 'para-faça', 'repetir-até' e outras estruturas de programação, requer a terminação de instruções, e implementa as mesmas estruturas de dados que estão disponíveis na maioria das linguagens de programação como C, C++ ou Java.
Esse é um exemplo de um programa simples escrito na linguagem de programação do MUPAD. Ele converte um inteiro decimal para sua forma binária. O exemplo mostra o uso de um operador de atribuição padrão com um 'repetir-até' volta. A linha debaixo do programa é o resultado de sua execução.

```
n_real:=128:
n_bin:="":
repeat
  digit := (n_real mod 2);
  digit_bin:=expr2text(digit);
  n_bin:=digit_bin.n_bin;
  n_real := (n_real div 2)
until n_real = 0 end_repeat:
print(n_bin):

"10000000"
```

A conversão do inteiro 128 resultou no binário 10000000. Um exemplo simples da utilização de programação no MUPAD. Uma vantagem adicional de MUPAD é sua simplicidade em produzir gráficos. Isto pode ser muito importante em cursos onde gráficos tem um papel de destaque, como por exemplo, no cálculo, na geometria analítica, e assim por diante. CAPITULO II

3. Conceitos Básicos no MUPAD

Anteriormente foi mencionado que a maioria dos sistemas de álgebra computacional pode ser utilizada de maneira interativa. Isso significa que é possível, por exemplo, digitar uma instrução para multiplicar dois números e esperar até que o MUPAD processe o resultado e o retorne na tela.
O MUPAD contem um sistema de ajuda(help) que permite efetuar consultas durante sua utilização. Com esse sistema de ajuda é possível encontrar detalhes sobre funções do sistema, informações sobre a sintaxe dessas funções, os seus parâmetros, etc. O conhecimento para utilizar o MUPAD como uma “calculadora de bolso inteligente” é quase intuitivo.
Após a inicialização do MUPAD, ele fica aguardando a entrada de comandos na sua “interface gráfica de usuário”. Em Sistemas Windows ou Macintosh o sinal ● representa a espera de comandos. Para facilitar a referencia no decorrer desse será definido que “Prompt” é a tela de espera de comando, ou seja, a Interface gráfica de usuário. O <ENTER> é interpretado como final da entrada de instruções e o sinal para sua execução. Mantendo a tecla <SHIFT> pressionada enquanto se utiliza o <ENTER>, não ocorre à execução das instruções, e sim, um avanço de linha. Em todas as versões é possível configurar essas teclas.
Ao digitar a instrução sin(3.141) no prompt e em seguida pressionar <ENTER> o resultado. 0.0005926535551 e exibido como resposta do sistema. O sistema avalia a função seno no ponto 3.141 e retorna uma aproximação de ponto flutuante do valor, funcionando de maneira semelhante uma calculadora eletrônica de bolso.
No MUPAD, uma variável global chamada “DIGITS” controla a precisão de ponto flutuante. Ao entrar com o comando “DIGITS := 100”, o MUPAD passara a executar todos os cálculos de ponto flutuante com uma precisão de 100 dígitos ou casas decimais.
Para finalizar uma instrução, deve inserir um terminador. Eles pode ser o dois-pontos “:”, que permite que o MUPAD execute o comando sem exibir o seu resultado na tela. Isto o permite suprimir a produção de resultados de intermediários irrelevantes. É permitido que duas ou

mais intruções sejam digitadas na mesma. Outro terminador é o ";". Dois comandos subseqüentes têm que ser separados por um ponto-e-vírgula:

diff(sin(x^2), x); int(last(1), x)

$$2 \cdot x \cdot \cos\left(x^2\right)$$

$$\sin\left(x^2\right)$$

Aqui x^2 representa o quadrado de x, enquanto as funções MUPAD "diff" e "int" representam, respectivamente, as operações de "derivada" e "integral". A função "last(1)" retorna a expressão anterior (no exemplo, este é a derivada de sin(x^2)).

No exemplo seguinte, a exibição do resultado do primeiro comando é suprimida pelo dois-pontos, e apenas o resultado do segundo comando se aparece na tela:

equation := {x + y = 1, x - y = 1}:

solve(equation)

$$\{[x = 1, y = 0]\}$$

No exemplo acima, um conjunto de duas equações é atribuído ao identificador "equation". A função "solve" processa a solução.

Os comandos e funções padrões do MuPAD são escritos com letras minúsculas. O aplicativo é "keysensivite", ou seja, diferencia letras maiúsculas e minúsculas. Por exemplo, sin(X) é diferente de Sin(X) (este comando não será reconhecido como função seno) ou sin(x).

Durante essa dissertação serão apresentadas diversas funções, assim como estruturas e operadores de programação, visando ter o conhecimento básico da linguagem de programação MUPAD, e assim, chegar aos exemplos de utilização no estudo do calculo.

4. Linguagem de programação do MUPAD

4.1. Conceitos básicos de programação

Uma Linguagem de programação é um conjunto de regras de sintaxe com os quais escrevemos listas de instruções. O MUPAD possui uma linguagem de programação de alto nível, ou seja, se assemelha a nossa forma de pensar e escrever. As Tarefas a serem realizadas em um programa MUPAD podem ser divididas em uma serie de instruções.

Variáveis e Tipos de Dados

Variáveis são objetos que armazenam dados e que, ao longo do tempo de execução de um programa MUPAD, podem ter seus valores alterados. Os nomes devem obedecer às regras de formação de identificadores. A atribuição de valores a uma variável é feita, com o operador de atribuição ":=".

Operações primitivas ou fundamentais

As Operações primitivas são apenas sete. No MUPAD elas possuem operadores e funções análogos, que são descriminados a seguir:

Atribuir: Consiste em fazer com que um dado assuma o valor de outro, de uma constante ou do resultado de uma expressão. Na linguagem de programação MUPAD a atribuição é feita com o símbolo dois pontos seguido do símbolo de igualdade, :=.

Comparar: Consiste em comparar valores quantitativamente ou a sua equivalência. Os símbolos matemáticos de igualdade =, e maior e menor >, < assim como suas combinações são validos na linguagem de programação MUPAD.

Armazenar: Consiste em armazenar dados nos dispositivos de armazenamento permanente. A função "write" armazena valores de identificadores em um arquivo, permitindo assim que os resultados computados possam ser utilizados posteriormente. O Exemplo a seguir demonstra a utilização da função "write" para armazenar os valores dos identificadores a e b para o arquivo "ab.mb":

a := 2/3: b := diff(sin(cos(x)), x):

write("ab.mb", a, b)

O Parâmetro inicial da função Write é o nome do arquivo de armazenamento. O MUPAD cria um arquivo com este nome no formato binário específico. Por conveção, um arquivo neste formato gerado pelo MUPAD deve ter a extensão ".mb". Utilizando a opção "Text" o MUPAD cria um arquivo em um formato de texto legível:

write(Text, "ab.mu", a, b)

A utilização da função write pode ser utilizada sem a expecificação dos identificadores a serem salvos. Nesse caso o MUPAD armazenara todos os identificadores disponíveis.

Recuperar: Consiste em retornar os dados armazenados permitindo assim que eles sejam reutilizados. No MUPAD a função read recupera os valores dos identificadores armazenados pela função write:

a := 1: b := 2: read("ab.mu"): a, b

$$\frac{2}{3}, -\sin(x) \cdot \cos(\cos(x))$$

No exemplo anterior a função read recupera o valor dos identificadores a e b que foram armazenados com a função write no arquivo "ab.mu".

4.2. Expressões e Operadores

O MUPAD contem uma grande quantidade de funções utilizadas para combinar ou manipular objetos. Porém, seria pouco intuitivo usar função _plus(a,b) para calcular a adição "a + b". Desse modo, o MUPAD implementa uma variedade de operações importantes, permitindo assim a utilização de notação matemática familiar.

Os operadores +, -, *, / para as operações aritméticas básicas e o operador ^ para a exponenciação são válidos para expressões simbólicas assim como suas funções equivalentes.

Operadores:
a + b + c, a - b, a*b*c, a/b, a^b
Funções:
_plus(a,b,c), _subtract(a,b),
_mult(a,b,c),_divide(a,b), _power(a,b)

A mesma equivalência existe para a função fatorial. A notação matemática n! é convertida internamente a uma chamada a função fact(n). Os operadores de um modo em geral podem ser utilizados em um contexto simbólico, entretanto, nesses casos, só retornam resultados simbólicos. Vários objetos de MUPAD separados por vírgulas formam uma seqüência:
sequencia := a, b, c + d

a, b, c + d

FALSO. A linguagem de programação MUPAD permite a combinação de expressões booleanas com os operadores lógicos and, or, ou o operador de negação utilizando o not. Na Linguagem de programação do MUPAD a definição de funções pode ocorrer de vários modos. O método mais simples é usar o operador "->" (o menos símbolo seguido pelo "maior que" símbolo):
f := x -> x^2

$$x \to x^2$$

desigualdade <>, respectivamente. Os operadores <, <=, > e >= comparam as magnitudes dos argumentos. O retorno dessas comparações dentro de um contexto comcreto são objetos do tipo boolean, ou seja, VERDADEIRO ou
Após a definição dessa função ela pode ser utilizada como uma função do MUPAD:
f(4), f(x + 1), f(y)

$$16, (x+1)^2, y^2$$

O MUPAD permite a definição de funções compostas. Essa Definição é efetuada por meio do operador "@".
c := a@b: c(x)

$$a(b(x))$$

Ou de forma simbólica:
x^i $i = 1..5

x, x^2, x^3, x^4, x^5

4, 9, 16, 25, 36, 49

Equações e desigualdades são objetos MUPAD válidos. Eles são gerados pelo sinal de igualdade = e pelo sinal de
A repetição do operador "@" duas vezes, denota a composição iterada de uma função com si mesmo um determinado numero de vezes:
f := g@@4: f(x)

$$g(g(g(g(x))))$$

O operador "$" é uma ferramenta importante para gerar tais sucessões:
i^2 $i = 2..7
O resultado da execução será:
Algumas funções MUPAD, como a função de integral definida necessita de um intervalo numerico. A geração desse intervalo é efetuada por meio do operador "..":
gama := 0..1; int(x, x = gama)

$$0..1$$

$$\frac{1}{2}$$

A tabela 1 contém os operadores da linguagem de programação do MUPAD e suas funções equivalentes.

Tabela 1 – Operadores da Linguagem de Programação do MUPAD

Operador	Função	Operação	Exemplo
+	_plus	Adição	SUM := a+b
-	_subtract	Subtração	Difference := a-b
*	_mult	Multiplicação	Product := a*b
/	_divide	Divisão	Quotient := a/b
^	_power	Exponeciação	Power := a^b
div	_div	Divisão Inteira	Quotient := a div p
mod	_mod	Resto da Divisão	Remainder := a mod p
!	fact	fatorial	n!
$	_seqgen	geração de sequência	Sequence := i^2 $ i=3..5
,	_exprseq	sequência	Seq := Seq1,Seq2
union	_union	União de conjuntos	S := Set1 union Set2
intersect	_intersect	Interceção de conjuntos	S := Set1 intersect Set2
minus	_minus	Diferença de conjuntos	S := Set1 minus Set2
=	_equal	Equação	Equation := x+y=2
<>	_unequal	Inequação	Condition := x<>y
<	_less	comparação	Condition := a<b
>		Comparação	Condition := a>b
<=	_leequal	Comparação	Condition := a<=b
>=		Comparação	Condition := a>=b
not	_not	Negação	Condition2 := not Condition1
and	_and	"e" lógico	Condition := a<b and b<c
or	_or	"Ou" lógico	Condition := a<b or b<c
->		Mapeamento de Função	Square := x -> (x^2)
'	D	Differential	f'(x)
@	_fconcat	composição de funções	h := f@g
@@	_fnest	Iteração	g := f@@3
.	_concat	Concatenação	NewName := Name1.Name2
..	_range	Variação	Range := a..b
...	interval	Intervalo	IV := 2.1 ... 3.5

4.3. Estruturas básicas

As operações primitivas e expressões podem ser agrupadas, formando conjuntos chamados estruturas básicas. Estruturas básicas podem ser classificadas de três modos: Estrutura de Seqüência, Estrutura de Seleção e

Estrutura de repetição.
Estrutura de Seqüência: Determina que as expressões e funções serão executadas seqüencialmente na ordem que foram escritas, isto é, da esquerda para direita de cima para baixo. A Linguagem MUPAD não possui um delimitador de inicio e final para a estrutura de seqüência. Essa estrutura esta implícita no interpretador da linguagem.

Estrutura de Seleção: Traduzem as escolhas entre caminhos alternativos, para a execução expressões e funções. No MUPAD a estrutura de seleção mais simples e o "IF". Essa estrutura avalia uma expressão lógica, cujo resultado deve ser booleano, ou seja, VERDADEIRO ou FALSO. Avaliando o resultado uma e somente uma das alternativas será executada. É possível também que o resultado gerado pelo caminho lógico seja apenas um desvio na estrutura do programa, ou seja não possua nenhuma instrução para ser executada. O final da estrutura de seleção "IF" e marcado pela ´palavra reservada "END_IF". A estrutura toda é compostas das. Palavras reservadas "IF", "THEN" que sinaliza o inicio da seqüência de instruções caso a expressão retorne VERDADEIRO, "ELSE" que sinaliza a seqüência de instruções caso a expressão retorne FALSO. A Estrutura testa a variável i, verificando se ela é um numero primo e a seguir exibe uma mensagem com essa informação:
if isprime(i) then print(expr2text(i)." é primo") else print(expr2text(i)." Não é primo") end_if:

"5 é primo"

é possível a inserção de outra estrutura "if" no interior de uma estrutura já definida, e no caso de estruturas aninhadas "se...outro..se" o MUPAD provê a palavra reservada "ELIF" que é uma abreviação de "ELSE IF". No caso do aninhamento de varias estruturas "ELIF" existe a estrutura alternativa "CASE". A estrutura "CASE" tem o final marcado pela estrutura "END CASE". O MUPAD avalia a expressão de teste da estrutura "CASE", comparando-a com os resultados apontados nas estruturas "OF...DO". Tendo uma comparação positiva o MUPAD executa todas as instruções subseqüentes até encontrar o delimitador de final "END CASE" ou a palavra reservada "BREAK". A Estrutura "CASE" ainda possui a opção "OTHERWISE" que é executada caso nenhuma das estruturas "OF..DO" anteriores a ela sejam satisfeitas.

```
case domtype(y)
  of DOM_INT do
  of DOM_RAT do
  of DOM_FLOAT do
    if y > 0 then y else -y end_if;
    break;
  of DOM_COMPLEX do
    sqrt(Re(y)^2 + Im(y)^2);
    break;
  otherwise
    "Invalid argument type";
  end_case:
```

Estrutura de Repetição: Permite que um grupo de instruções possa ser executado um número variável de vezes. Para cada repetição dos comandos é feito um teste lógico que ira determinar a execução dos comandos ou o fim da repetição. No MUPAD a estrutura "PARA" é representada pela palavra reservada "FOR":

```
for i from 1 to 4 do
     x := i^2;
     print("O Quadrado de ", i, " eh ", x)
  end_for:
```

"O Quadrado de ", 1, " eh ", 1

"O Quadrado de ", 2, " eh ", 4

"O Quadrado de ", 3, " eh ", 9

"O Quadrado de ", 4, " eh ", 16

O exemplo anterior demonstra a utilização do FOR para a exibição dos quadrados dos números inteiros de 1 a 4. A variável i recebe os valores 1, 2, 3 e 4 automaticamente e todas as instruções entre o "FOR" e o "END_FOR" são executadas. Para fazer uma repetição utilizando uma variável que decrescesse utilizaria se a palavra reservada "DOWNTO":

```
for j from 4 downto 2 do
     print(Unquoted,
          "O Quadrado de ".expr2text(j)." eh ".
          expr2text(j^2))
  end_for:
```

O Quadrado de 4 eh 16

O Quadrado de 3 eh 9

O Quadrado de 2 eh 4

A palavra reservada "STEP" permite incrementos ou decrementos diferentes de 1 e a palavra reservada "NEXT" permite que a estrutura vá para a próxima iteração sem ter que executar as expressões restantes no bloco de código.

```
for i from 10 downto -10 step 2 do
     if i < 0 then next end_if;
     x := sqrt(i);
     print(x)
  end_for:
```

O resultado da execução é:

$$10$$

$$2^{1/2}\,2^{2}$$

$$6^{1/2}$$

$$2$$

$$2^{1/2}$$

$$0$$

Para repetições que são controladas por expressões lógicas o MUPAD possui duas estruturas a "REPETIR" e a "ENQUANTO", representadas respectivamente por, "REPEAT" e "WHILE". A Estrutura "REPEAT" tem como finalizador a palavra reservada "END_REPEAT". A Condição de repetição é sinalizada com a palavra "UNTIL" e deve ser colocada antes da finalização do bloco, ou seja logo antes do "END_REPEAT":

```
x := 2:

repeat
      i := x; x := i^2; print(i, x)
  until x > 100 end_repeat:
```

O resultado da execução será:

$$2,\ 4$$

$$4,\ 16$$

$$16,\ 256$$

Na Estrutura "REPEAT" a verificação da condição de repetição é feita após a execução do bloco de instruções. Quando a condição se torna FALSA a repetição é interrompida. É possível interromper a repetição também com o sinalizador de interrupção "BREAK". A Estrutura "WHILE" funciona de forma análoga a estrutura "REPEAT", porem, a condição é testada antes do bloco de instruções.

```
x := 2:
while x <= 100 do
      i := x; x := i^2; print(i, x)
  end_while:
```

O resultado da execução será:

$$2,\ 4$$

$$4,\ 16$$

$$16,\ 256$$

4.4. A biblioteca padrão

A biblioteca mais importante do MUPAD é a stdlib a bibliotecay padrão. Ela Contém as funções que são freqüentemente utilizadas como diff e simplify alem de uma grande quantidade de constantes e funções matemáticas (Tabelas 2 e 3).

Tabela 2 – Funções da Linguagem de Programação do MUPAD

FUNÇÕES		
abs(x)	$\lvert x \rvert$	Valor absoluto
exp(x)	e^x	Exponencial
ln(x)	ln(x)	Logaritimo
sign(x)	$\frac{x}{\lvert x \rvert}$	
sqrt(x)	$\sqrt{x}$	Raiz Quadrada
cos(x)	cos(x)	Coseno
sin(x)	sen(x)	Seno
tan(x)	tg(x)	Tangente
acos(x)	arccos(x)	Arco Coseno
asin(x)	arcsen(x)	Arco Seno
atan(x)	arctg(x)	Arco tangente

Tabela 3 – Constantes da Linguagem de Programação do MUPAD

Constantes		
E	2.7182818...	Numero de Euller
infinity	∞	Infinito Positivo
PI	π	Valor
undefined		Valor Indefinido

Todas as funções da stdlib estão automaticamente disponíveis ao iniciar a utilização do MUPAD. Não há nenhuma diferença perceptível entre a performance das funções da biblioteca padrão e das funções implementadas na linguagem de programação do MUPAD.

4.5. MUPAD PROCEDURES

O MUPAD permite Construções essenciais a uma linguagem de Programação. O Usuário pode implementar algoritmos complexos confortavelmente com o MUPAD. A maioria da inteligência matemática do MUPAD não é implementada em C ou C++ dentro do Kernel (núcleo) do Software, mas na linguagem de programação nativa do MUPAD, ao nível de biblioteca. Com o conhecimento das estruturas de repetição e seleção e de funções "simples" é possível iniciar a programação de Procedures mais complexas. Tais procedures são geradas através ***proc – end proc***. E neles veremos que a distinção entre variáveis locais e globais. Após a Procedure estar pronta, veremos que é possível chamá-la no prompt como uma outra função qualquer. Normalmente o usuário deseja utilizar estes procedimentos diversas vezes e em sessões posteriores, em particular, quando eles implementam algoritmos mais complexos. É de grande utilidade escrever a definição da procedure em um arquivo texto e utilizar isso no MUPAD através de uma simples leitura. A nível de avaliação, kernel do MUPAD processa os comandos precisamente, como se eles fossem digitados no Prompt do sistema.

A função a seguir, é um exemplo de uma definição de procedimento por ***proc – end proc***. Ela compara dois números e retorna o máximo entre eles.

```
• Max := proc(a, b) /* comment: maximum of a and b */
        begin
          if a<b then return(b) else return(a) end_if
        end_proc:
```

O texto entre / * e * / é um comentário completamente ignorado pelo sistema. Esta é uma ferramenta útil para documentar o código fonte quando se defini uma procedure em um arquivo texto. Uma Procedure gerada por ***proc – end proc*** é do tipo DOM_PROC:

```
• domtype(Max)
```

DOM_PROC

Utilizando o identificador que foi associado à procedure, você pode utilizá-la como uma função MUPAD:

```
• Max(1/5,0.5)
```

0.5

Dentro de uma Procedure você pode chamar funções de sistema ou outras procedures, e ainda implementar algoritmos de recursividade. O exemplo favorito para um algoritmo de recursividade é o fatorial, uma função de números naturais(inteiros positivos) que pode ser definida como n! = 1. 2.. n. ou n! = n. (n - 1)! sabendo que por definição 0! = 1. A Seguir temos 2 maneiras de se calcular um fatorial. A primeira através de uma repetição simples e a segunda com recursividade:

```
• fatorial_1 := proc(n) begin
      fat := 1;
      for i from 1 to n do
          fat := fat * i;
      end_for;
      return(fat)
      end_proc:
• fatorial_2 := proc(n) begin
      if n = 0 then
         return(1)
      else return(n*fatorial_2(n - 1))
      end_if
      end_proc:
```

No MUPAD a Variável de ambiente MAXDEPTH determina o numero Maximo de iterações, ou seja o numero de vezes que uma procedure pode chamar outra ou ela mesma. Por padrão MAXDEPTH tem o valor 500. isto é, a função Fatorial_2 pode calcular fatoriais para n <= 500. Valores de n Maiores que 500 interrompem a função e causam mensagens de erro. Incrementando MAXDEPTH é possível se calcular fatoriais maiores.

A chamada de uma procedure, executa seu corpo, isto é, a sucessão de declarações entre ***begin*** e o ***end_proc***. Toda procedure devolve algum valor, explicitamente por retorno através do comando ***return*** ou na sua ausência o valor do último comando executado dentro da procedure. Assim, a função fatorial_2 poderia ter sido implementada sem o **return**.

```
• fatorial_2 := proc(n) begin
      if n = 0 then 1 else n*fatorial_2(n - 1) end_if
        end_proc:
```

Uma Procedure pode retornar um objeto MUPAD, uma expressão, uma seqüência, um conjunto, ou até mesmo uma Procedure. Contudo se essa procedure utilizar variáveis locais de fora da procedure, ela deve ser definida coma a opção ***escape***. Do contrário, essa procedure conduz o MUPAD a exibir mensagens de advertências ou produzir efeitos indesejados. A procedure seguinte retorna uma função que utiliza o parâmetro power fora da Procedure.

```
• generatePowerFunction := proc(power)
        option escape;
        begin
          x -> (x^power)
        end_proc:
• f := generatePowerFunction(2):
• g := generatePowerFunction(5):
• f(a), g(b)
```

$$a^2, b^5$$

Muitas funções do sistema têm retorno de funções simbólicas, devido ao MUPAD não encontrar uma representação mais simples para a requisição:

```
• sin(x), max(a, b), int(exp(x^3), x)
```

$$\sin(x), \max(a,b), \int e^{x^3}\,dx$$

O mesmo comportamento se repetirá em seus próprios procedimentos, quando você encapsular o nome de procedimento em um hold. O hold impede a função de se chamar recursivamente e terminar em uma recursão infinita. A função seguinte calcula o valor absoluto para valores numéricos (inteiros, números racionais, números reais de ponto flutuante, e números complexos). Para todos os outros tipos de valores, o ela possui um retorno simbólico:

```
• Abs := proc(x) begin
    if testtype(x, Type::Numeric) then
      if domtype(x) = DOM_COMPLEX then
          return(sqrt(Re(x)^2 + Im(x)^2))
      else if x >= 0 then
            return(x)
          else return(-x)
          end_if
      end_if
    end_if;
    hold(Abs)(x)
  end_proc:
```

- Abs(-1), Abs(-2/3), Abs(1.234), Abs(2 + I/3), Abs(x + 1)

$$1, \frac{2}{3}, 1.234, \frac{\sqrt{37}}{3}, Abs(x+1)$$

Uma maneira mais elegante para obter um retorno simbólico seria utilizando o Objeto MUPAD que devolve o nome do procedimento de chamada e a função *args()* que retorna a seqüência de parâmetros de entrada da procedure. Vejamos a seguir como ficaria a procedure Abs implementada nesses moldes.

```
• Abs := proc(x) begin
    if testtype(x, Type::Numeric) then
      if domtype(x) = DOM_COMPLEX then
          return(sqrt(Re(x)^2 + Im(x)^2))
      else if x >= 0 then
            return(x)
          else return(-x)
          end_if
      end_if
    end_if;
    procname(args())
  end_proc:
```

- Abs(-1), Abs(-2/3), Abs(1.234), Abs(2 + I/3), Abs(x + 1)

$$1, \frac{2}{3}, 1.234, \frac{\sqrt{37}}{3}, Abs(x+1)$$

A utilização de Variáveis em procedures é permitida. Essas variáveis externas são chamadas de variáveis globais. Observe o comportamento da Procedure abaixo:

```
• a := b: f := proc() begin a := 1 + a^2 end_proc:
```

- f(); f(); f()

$$b^2 + 1$$

$$\left(b^2+1\right)^2 + 1$$

$$\left(\left(b^2+1\right)^2+1\right)^2 + 1$$

A procedure ***f*** altera o valor de uma variável que foi definida fora da procedure. Quando a procedure termina, a variável ***a*** um tem um valor novo que é alterado novamente através de chamadas adicionais a procedure ***f***.

Utilizando a Palavra-Chave ***Local*** os identificadores são declarados como variáveis locais e só são validos dentro do procedimento que as declarou.

```
• a := b: f := proc() local a; begin a := 2 end_proc:
```

- f():a

$$b$$

Apesar dos nomes iguais das variáveis, a atribuição ***a*** := 2 na variável local, não afeta o valor do identificador global ***a*** definida fora da procedure. Não existe limite para o numero de variáveis locais declaradas em um procedimento. Basta declará-las apos a palavra chave ***local***.

5. Conclusão

O uso de da linguagem de Programação incluida no software Mupad no processo de ensino-aprendizagem permitiu o desenvolvimento de diversas soluções de um mesmo problema possibilitado o estudante abordar problemas complexos com maior facilidade. Não obstante das soluções numéricas, procurou-se tornar a Matemática mais experimental, e analisar diferentes situações com visualização gráfica, dando assim oportunidade ao estudante de aprender fazendo

Por outro lado, o uso de softwares do tipo “freeware” com funcionalidaes similades ao dos softwares pagos desetimula o uso de cópias nào autorizadas.

Sugere-se a continuidade do trabalho conjunto entre a área de Informática e a área de Matemática na procura de metodologias que incentivem e facilitem o aprendizado das Ciencias Exatas.

BIBLIOGRAFIA

[1] Castaño, Carlos. A pesquisa nos meios e materiais de ensino. In: SANCHO, Juana Maria (org). Para uma tecnologia educacional. Porto Alegre: Artmed, 2001.

[2] Drescher,K et al. MuPAD User's Manual and CD-ROM: Multiprocessing Algebra Data Tool, MuPad Version 1.2.2.- Mupad Gruop (Corporate Author)