MODELOS COMPUTACIONAIS PARA VERIFICAÇÃO DE IDENTIDADES POLINOMIAIS EM ÁLGEBRAS

Márcio Pedro dos Santos Sousa[1]; Francisco Bruno Souza Oliveira[2]
[1]Mestrando em Universidade Estadual de Santa Cruz; Bolsista; PD&I HPC–FAPESB;marcio5858@hotmail.com
Universidade Estadual de Santa Cruz – UESC Ilhéus - BA; fbsoliveira@uesc.br

RESUMO

Nesse trabalho apresentamos uma abordagem computacional para tratar das álgebras que satisfazem identidades polinomiais. Mais precisamente, utilizamos a linguagem python para verificar e identificar a identidade polinomiais da álgebra de Grassmann e a de Hall . Foram criados alguns procedimentos para adequar o produto das matrizes segundo as propriedades da algebra, sendo esta uma álgebra não comutativa. Este procedimento apresenta algumas funções previamente implementadas que permitem trabalhar com tais propriedades, porém, o tempo de processamento é consideravelmente maior em comparação com algumas das funções que implementamos. Finalizamos com os cálculos explícitos das operações dos elementos das matrizes.

PALAVRAS-CHAVE: : PI-Álgebra; Identidade polinomial; Grassmann

1. INTRODUÇÃO

Nesse trabalho buscamos uma abordagem computacional para obter e abrir caminho para um processo menos limitado pela ferramenta ou lingguagem utilizada, tendo em vista que este trabalho teve como um dos seus objetivos aprimorar e mostrar certas limitações de trabalhos encontrados na literatura. Mais precisamente, utilizaremos linguagem Python, para a verificação e identificação se é, ou não uma identidade polinomial em questão. Em especial temos as identidades polinomiais das álgebras de matrizes com entradas na álgebra de Grasmann E. Finalizaremos o trabalho com alguns casos explícitos e o tempo que o processo leva para calcular.

2. METODOLOGIA

A pesquisa foi dividida em três partes principais, sendo revisão da literatura, desenvolvimento matemático da teoria, e modelagem utilizando linguagem python.

A revisão da literatura destinou-se à pesquisa dos métodos clássicos de obtenção das identidades polinomiais e suas técnicas desenvolvidas e das técnicas desenvolvidas por colaboradores e grupos de pesquisas das instituições parceiras. Foi utilizado levantamentos de dados obtidos por outras ferramentas computacionais como base de aprimoração dos resultados. Foram observados em outras ferramentas limitações da própria ferramenta utilizada para a elaboração dos processos, sendo uma dessas o Maple

Para o desenvolvimento matemático, o primeiro momento da pesquisa buscou-se encontrar um algoritmo que incorporasse as técnicas principais, sendo em um posterior segundo momento a otimização deste com técnicas levantadas das próprias bibliotecas do python. Foi considerado inicialmente o uso de funções já contidas nessa linguagem com o propósito de testa já a eficiência dos processos apenas na mudança de linguagem e ferramenta. Para o segundo momento, foram feitas adaptações para aprimorar um pouco mais o modo a se implementar os métodos de otimização.

3. RESULTADOS E DISCUSSÃO

Iniciamos a construção dos algoritmos fazendo uso da classe denominada Commutator, que calcula o comutador entre duas variáveis. Em notação matemática, o comutador de duas variáveis x, y é dado por [x, y] = xy - yx . Vale lembrar que são usados apenas dois parâmetros em cada função, podemos verificar a identidade da seguinte forma:

```
from sympy.physics.quantum import Commutator, Operator,AntiCommutator

from sympy.abc import x, y

from sympy import Symbol
```

```
x1 = Symbol('x1')

x2 = Symbol('x2')

x3 = Symbol('x3')

F1 = Commutator(x1, x2)

F2 = Commutator(F1, x3)

F2.doit()
```

Onde foi obtido o valor “zero”, como esperado. Porém, a função Commutator mostra que a mesma não é uma boa opção para o simples cálculo de comutadores de comprimento bem maior, quando as variáveis são não-comutativas. Por conta disso, apresentaremos alguns procedimentos para resolver esse e outros problemas, como por exemplo, a regra do produto da álgebra de Grassmann, isto é, xy = -yx. Para este procedimento, declaramos o tipo de entrada de dados (list) e usamos a variável i como controlador de *loop*, enquanto *dstr* armazena a expressão de saída. Usando esse procedimento podemos dar entrada com uma lista de tamanho n maior que 2 e obtemos uma expressão dada pelo colchete de Lie. Porém, apesar de calcular de forma mais rápida, a função comutador não leva em consideração nenhuma propriedade da variável. Ou seja, caso queiramos que as variáveis anti-comute, precisamos criar novos procedimentos, os quais descrevemos na pesquisa.

4. CONSIDERAÇÕES FINAIS

Após o estudo e a elaboração do caminho para modelar em python a obtenção sobre as álgebras com identidades polinomiais, começamos os procedimentos de modelos computacionais para trabalhar com álgebras não comutativas. Dentre esses modelos, construímos uma forma mais abrangente e menos limitada para obtenção de resultados mais abrangente, com uso da linguagem python, principalmente para calcular permutações de matrizes com *n* maiores do que os encontrados na literatura. Bem como para satisfazer propriedades de álgebras específicas tais como a álgebra de Gassmann e a identidade de Hall. Vale ressaltar o uso do colchete de lie para a obtenção da verificação da identidade de Hall. Para tal, foi criada uma função denominada HALL, tal que assim como para a álgebra de grassmann também se implementa o colchete de lie. Assim obtivemos a seguinte função.

Calculamos também a nível de curiosidade o tempo levado para obtenção dos cálculos das multiplicações, usando a função *time.*

```
ini = time()
J = HALL(A,B,C)
fim = time()
print(fim-ini)
0,04505658149719238
ini = time()
X = comutador(A,B) \\ [X1, X2]
Y = np.dot(X,X) \\ [X1,X2] [X1,X2]
Z = comutador(Y,C) \\ [[X1,X2] ,X3]
fim = time()
print(fim-ini)
0,021277666091918945
```

Comparando com a função *comutador* do *python* vemos que não temos uma eficiência em relação ao tempo de processamento, em quanto ao uso da função HALL, o que confirma ainda mais o uso da linguagem python como ferramenta essencial para o estudo de novas identidades polinomiais e suas aplicações. Vale lembrar que por conta do momento que estamos vivendo e das limitações imposta pelo tempo, deixamos como trabalho futuro implementar mais identidades e verificações do total potencial dessa linguagem na PI-Álgebra.

5. REFERÊNCIAS

[1]MONTEIRO, R. S.**Modelos Computacionais para Verificação de Identidades Poli-nomiais em Álgebras dos Tipos Mn(E) e Mk,l(E)**. Dissertação (Programa de Pós-graduação em Modelagem Computacional) - Universidade Estadual de Santa Cruz.Bahia

[2]ZIMMERMANN, H.**PI-Algebras: An Introduction**.Berlin - New York: Lectures Notesin Math., Springer-Verlag, 1975.

[3]ALVES, S. M.; SARTORI, K. K.; ARAKAWA, V. **A.The Standard Identity on Mn(E) in Characteristic p >2**. Journal of Algebra, v. 9, n. 4, p. 155–158, 2015.

[4]BENTIN, R. M.; ALVES, S. M.Building Grassmann **Numbers from PI-Algebras.ArXive-prints**, abr, 2012.

[5]AMITSUR, S. A.; LEVITZKI, J.**Minimal Identities for Algebras. Math**. Soc. 1, p.449–463, 1950.

[6]AZEVEDO, S. S.**Identidades Graduadas para Álgebras de Matrizes.Dissertação** (Tese de Doutorado) — Instituto de Matemática, Estatística e Computação Científica (UNI-CAMP), 2003.

[7]AMITSUR, S. A.; LEVITZKI, J.**Minimal Identities for Algebras.Math**. Soc. 1, p.449–463, 1950.

[8]BENTIN, R. M.; ALVES, S. M. **Building Grassmann Numbers from PI-Algebras**.ArXiveprints,abr, 2012