PROFESSOR DE MATEMÁTICA ONLINE
Revista eletrônica da Sociedade Brasileira de Matemática

PMO v.12, n.3, 2024
ISSN: 2319-023X
https://doi.org/10.21711/2319023x2024/pmo1225

Criação de tecnologias com o *software SageMath* para teoria dos números

Ricardo N. Machado Junior Thiago Yukio Tanaka Jogli Gidel da Silva Araújo

Resumo

Neste artigo, vamos explorar conceitos da Teoria dos Números e, em paralelo, desenvolver tecnologias computacionais no intuito de tornar mais acessível e dinâmico os estudos e pesquisas neste tema. Por meio do *software SageMath*, uma ferramenta matemática de código aberto integrada ao Python, o objetivo é criar tecnologias para o ensino dos conceitos fundamentais da Teoria dos Números que possibilitem um melhor entendimento global no tema, por meio da visualização geométrica, conferência de cálculos intermediários ou totais e determinação dos casos de solução. Exploramos temas como o Algoritmo da Divisão Euclidiana, o Máximo Divisor Comum (MDC), Equações Diofantinas e o Número de Frobenius, e oferecemos uma gama de abordagens didáticas, com disponibilidade de código fonte e *widgets* acessíveis *online*, visando auxiliar professores e estudantes nas questões de ensino e aprendizagem no tema.

Palavras-chave: Máximo Divisor Comum; Equações Diofantinas Lineares; Número de Frobenius; Algoritmo; *SageMath*.

Abstract

In this article, we will explore concepts from Number Theory and, in parallel, develop computational technologies in order to make studies and research on this topic more accessible and dynamic. Through *software SageMath*, an open source mathematical tool integrated with Python, the objective is to create technologies for teaching the fundamental concepts of Number Theory that enable a better global understanding of the topic, through geometric visualization, checking intermediate or total calculations and determining solution cases. We explore topics such as the Euclidean Division Algorithm, the Greatest Common Divisor (GCD), Diophantine Equations and the Frobenius Number, and offer a range of teaching approaches, with source code availability and accessible *widgets online* , aiming to assist teachers and students in teaching and learning issues on the topic.

Keywords: Greatest Common Divisor; Linear Diophantine Equations; Frobenius Number; Algorithm; *SageMath*.

1. Introdução

A Base Nacional Comum Curricular (BNCC) ressalta a necessidade de preparar as crianças e adolescentes para acompanhar os avanços da computação e das tecnologias digitais. Este trabalho estabelece uma conexão entre a elementos da Teoria dos Números, Computação e as tecnologias digitais. Essa interligação é fundamentada e incentivada pela BNCC por meio dos seguintes trechos:

SBM SOCIEDADE BRASILEIRA DE MATEMÁTICA

> *A BNCC propõe que os estudantes utilizem tecnologias, como calculadoras e planilhas eletrônicas, desde os anos iniciais do Ensino Fundamental. Essa valorização possibilita que, ao chegarem aos anos finais, eles possam ser estimulados a desenvolver o pensamento computacional, por meio da interpretação e da elaboração de algoritmos, incluindo aqueles que podem ser representados por fluxogramas.* (BNCC p. 528, [8])

> *...destaca-se ainda a importância do recurso a tecnologias digitais e aplicativos tanto para a investigação matemática como para dar continuidade ao desenvolvimento do pensamento computacional.* (BNCC p. 528, [8])

O *SageMath* surge como uma ferramenta matemática nesse contexto, caracterizando-se por ser de código aberto (*open source*) e integrar o Python. Isso possibilita a execução de qualquer código ou biblioteca para Python diretamente no *SageMath*, tornando-o uma escolha ideal para estudantes, professores e pesquisadores de matemática (conforme detalhado em [6]). Além disso, o *SageMath* oferece flexibilidade ao ser instalado em computadores ou utilizado gratuitamente na nuvem, por meio da plataforma CoCalc (https://cocalc.com/) ou diretamente no site https://sagecell.sagemath.org/. Essa acessibilidade é estendida a *smartphones*, *tablets* ou computadores conectados à internet, ampliando ainda mais seu alcance educacional (para mais informações sobre o *SageMath*, consulte [4] ou [7]).

O SageInteract, por sua vez, emerge como uma ferramenta complementar poderosa e versátil dentro do ecossistema *SageMath*. Desenvolvido para tornar a matemática interativa e acessível em todos os níveis de habilidade, o SageInteract permite a criação de documentos interativos que incluem gráficos, widgets[1] e simulações. Sua capacidade de incorporar widgets interativos diretamente nos cálculos matemáticos oferece uma abordagem intuitiva e prática, enriquecendo significativamente o processo de aprendizado ou de ensino (para mais informações sobre o SageInteract, consulte [5]).

Este trabalho tem como objetivo central o desenvolvimento, implementação e disponibilização de funções e widgets no *SageMath*, direcionados ao ensino dos conceitos fundamentais da Teoria dos Números. Apresentamos recursos que proporcionam uma variedade de abordagens didáticas, favorecendo a adaptabilidade nas aulas e promovendo metodologias ativas. Ao abordar temas como o Algoritmo da Divisão Euclidiana, Máximo Divisor Comum, Equações Diofantinas Lineares e o Número de Frobenius, buscamos oferecer uma contribuição significativa ao ensino e aprendizagem no tema. Por exemplo, há possibilidade de utilização por docentes do Ensino Superior nas disciplinas Aritmética dos Inteiros ou Teoria dos Números, possibilitando a criação de problemas de exames, conferência rápida dos cálculos, além de servir como suporte dentro da sala de aula. Isso se estende também no Ensino Básico uma vez que alguns dos temas listados fazem parte do currículo, além de serem centrais em treinamentos de Olimpíadas Matemáticas. Além disso, como os códigos estão disponibilizados neste artigo, eles podem ser copiados e modificados, permitindo a criação de outros contextos. Assim o artigo oferece possibilidades ilimitadas de abordagens didáticas direcionadas para a necessidade de cada usuário.

Os *widgets* desenvolvidos foram disponibilizados *online*, facilitando o acesso para qualquer leitor(a) interessado(a). Além disso, pode-se utilizá-los em dispositivos como computadores, *tablets* e *smartphones*. O material pode ser acessado através do *QR code* abaixo ou pelo seguinte endereço eletrônico:

[1] Um widget é um elemento de interface gráfica de usuário (GUI) que permite aos usuários interagir com um programa de computador. Em um contexto de programação, os widgets são utilizados para criar interfaces interativas, permitindo aos usuários manipular e visualizar dados de forma dinâmica.

https://prof-ricardomachado.github.io/artigo-frobenius/artigo-frobenius.html

Por fim, é válido mencionar que outras contribuições nessa linha, porém abordando diferentes tópicos, podem ser encontradas em referências como [10], [11] e [12].

2. Conceitos da Aritmética dos Inteiros

Nesta seção, vamos estabelecer alguns critérios sobre divisibilidade de números inteiros. Além disso, o nosso intuito é construir ferramentas para determinar o Máximo Divisor Comum entre dois números inteiros.

Definição 1. Sejam a e b números inteiros. Dizemos que a divide b no conjunto $\mathbb{Z}$ quando existe algum número inteiro c tal que $b = ca$, e denotamos por $a|b$. Quando $a|b$, b é um *múltiplo* de a.

Teorema 1 (Divisão Euclidiana)**.** *Sejam a e b números inteiros com $b \neq 0$. Então existem únicos inteiros q e r tais que*

$$a = bq + r, \quad com \quad 0 \leq r < |b|.$$

O número a é chamado de dividendo*, o número b é chamado de* divisor*, o número q é chamado de* quociente *e o número r é chamado de* resto*.*

Demonstração. Veja página 16 de [3] ou página 46 de [2]. □

Alternando com a parte computacional, disponibilizamos uma função para *SageMath* no Bloco de Códigos 1. A função definida com o nome `divisao` recebe duas entradas `a` e `b` e retorna o resultado da Divisão Euclideana, conforme o Teorema 1. Na linha 5, a função é chamada com as entradas `a=-7` e `b=-3` e na Saída do Bloco de Códigos 1 está o resultado, após executar o Bloco de Códigos 1. No *SageMath*, o caractere "#" serve para deixar comentários no código, usamos isto nas linhas 2, 3, e 4 para indicar o que foi implementado em cada uma dessas linhas.

Bloco de Códigos 1.

```
1 def divisao(a, b):
2     r=ZZ(mod(a,b))                        # cálculo do resto da divisão de a por b
3     q=(a-r)/b                             # cálculo do quociente da divisão de a por b
4     print('%d = %d*%d + %d' %(a,b,q,r))   # imprime na tela os valores de a=b*q+r
5 divisao(-7, -3)
```

Saída do Bloco de Códigos 1.

```
1  -7 = -3*3 + 2
```

Observação 1. É extremamente importante que ao utilizar os códigos deste artigo, toda a estrutura de indentação seja preservada, um aspecto intrínseco da programação em *SageMath*/Python. Por exemplo, apenas copiar e colar os códigos acima sem as indentações nas linhas 2, 3 e 4, o programa retornará uma mensagem de erro.

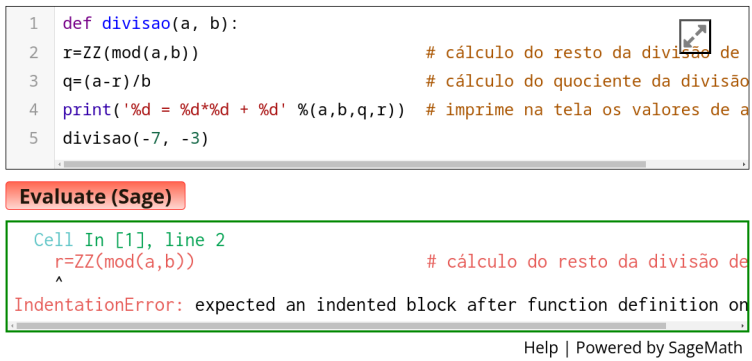


Figura 1: Código com identação incorreta retornará uma mensagem de erro.

A Figura 2 contém o código com a indentação correta.

Além disso, a cópia do texto do artigo por meio de seleção dos caracteres, "*control* + c"e a colagem no bloco de códigos usando "*control* + v", pode modificar caracteres especiais como o símbolo do circunflexo (ˆ). Verificamos este erro copiando os textos do artigo e colando-os diretamente no bloco de códigos para o exemplo desenvolvido no Bloco de Códigos 10.

No código anterior, o comando `ZZ(mod(a,b))` foi utilizado para que o *software* realize o cálculo correto da Divisão Euclidiana, independente dos valores inteiros serem positivos ou negativos. O comando `a%b` é mais utilizado para este cálculo, mas no *SageMath/Python*, se os valores de a ou b forem negativos, o "resto" encontrado nem sempre é positivo.

Nas figuras abaixo, temos os *prints* das duas versões da função definida no Bloco de Códigos 1, uma transformada em *widget*, pelo SageInteract, e a outra idêntica a que está no artigo. Ambas as versões estão disponíveis no complemento deste artigo citado na introdução. Além disso, todas as outras funções também estão disponíveis nos dois formatos.

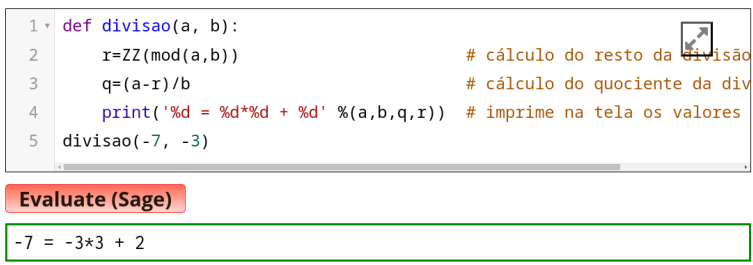


Figura 2: Código executável.

Tecnologia 1.2. Escolha valores para a e b para obter os valores do quociente e resto na divisão de a por b.

Figura 3: Função transformada em *widget* pelo SageInteract.

Definição 2 (Máximo Divisor Comum (MDC))**.** Sejam a e b dois inteiros, diz-se que o inteiro positivo d é o *Máximo Divisor Comum* de a e b, e denotamos por $d = \text{mdc}(a, b)$, se d satisfaz as seguintes condições:

(i) d é um divisor comum de a e b, ou seja, $d|a$ e $d|b$;

(ii) d é divisível por todo divisor comum de a e b, isto é, se c é divisor comum de a e b, então $c|d$.

Lema 1 (Euclides)**.** *Sejam* a, b, n *inteiros, então existe* $\text{mdc}(a, b)$ *e*

$$\text{mdc}(a, b) = \text{mdc}(a, b - an).$$

Demonstração. Veja página 75 de [2]. □

No *SageMath*, o método `gcd` calcula o MDC. (*gcd* é a abreviação para *Greatest Common Divisor*). No código abaixo, calculamos o MDC entre 25 e 88.

Bloco de Códigos 2.

```
gcd(25,88)          #calcula o MDC entre 25 e 88
```

Saída do Bloco de Códigos 2.

```
1
```

O Lema 1 também é útil para calcular o MDC entre dois valores que dependem de uma variável inteira, como pode ser visto no exemplo a seguir.

Exemplo 1 (*IMO* 1959, 1ª Edição, Problema 1)**.** Prove que

$$\frac{21n + 4}{14n + 3}$$

é irredutível para todo número natural n.

Solução. Para mostrar que $\frac{21n+4}{14n+3}$ é irredutível, basta mostrar que

$$\text{mdc}(21n + 4, 14n + 3) = 1, \quad \text{para todo } n \text{ inteiro.}$$

Usando que $\text{mdc}(a,b) = \text{mdc}(b,a)$ e aplicando o Lema 1, temos

$$\begin{aligned} \text{mdc}(21n+4, 14n+3) &= \text{mdc}(7n+1, 14n+3) \\ &= \text{mdc}(7n+1, 7n+2) \\ &= \text{mdc}(7n+1, 1) \\ &= 1. \end{aligned}$$

□

O *SageMath* também é capaz de calcular o MDC envolvendo variáveis como no Exemplo 1.

Bloco de Códigos 3.

```
n = polygen(ZZ,'n') #indeterminada n no anel dos polinômios com coeficientes inteiros
gcd(21*n+4, 14*n+3) #calcula o MDC entre 21n+4 e 14n+3
```

Saída do Bloco de Códigos 3.

```
1
```

Perceba que ao utilizar o Bloco de Código 3, fizemos uma conferência do resultado. Porém, ao modificar as entradas do método `gcd`, é possível criar uma série de problemas semelhantes, com o mesmo perfil e dificuldade ao problema que foi proposto na *IMO*. Isso abre margem para problemas que tenham MDC diferente de 1, por exemplo, permitindo uma abordagem didática diferenciada na criação de problemas.

Teorema 2 (Algoritmo de Euclides para calcular o MDC)**.** *Sejam a e b inteiros positivos. O* $\text{mdc}(a,b)$ *pode ser calculado com os seguintes passos.*

Passo 1: *Faça* $i = 1$, $r_0 = \max\{a,b\}$ *e* $r_1 = \min\{a,b\}$*;*

Passo 2: *Defina* r_{i+1} *como o resto da divisão de* r_{i-1} *por* r_i*, isto é,* $r_{i-1} = q_i r_i + r_{i+1}$ *com* $0 \leq r_{i+1} < r_i$*;*

Passo 3: *Se* $r_{i+1} \neq 0$ *incremente* i *de 1 e volte para o passo* ***Passo 2****, caso contrário* $\text{mdc}(a,b) = r_i$.

Demonstração. Veja página 17 de [3]. □

Exemplo 2. Vamos utilizar o Algoritmo de Euclides (Teorema 2) para calcular $\text{mdc}(88,25)$.

Solução. Defina $r_0 = 88$ e $r_1 = 25$. Efetuando a Divisão Euclidiana de r_0 por r_1 e seguindo os passos do algoritmo:

$$88 = 25 \times 3 + 13 : (r_0 = 88, r_1 = 25, r_2 = 13).$$

Como $r_2 = 13 \neq 0$, definimos $i = 2$ e voltamos para o **Passo 2**:

$$25 = 13 \times 1 + 12 : (r_1 = 25, r_2 = 13, r_3 = 12).$$

Como $r_3 = 12 \neq 0$, definimos $i = 3$ e voltamos para o **Passo 2**:

$$13 = 12 \times 1 + 1 : (r_2 = 13, r_3 = 12, r_4 = 1).$$

Como $r_4 = 1 \neq 0$, definimos $i = 4$ e voltamos para o **Passo 2**:

$$12 = 1 \times 12 + 0 : (r_3 = 12, r_4 = 1, r_5 = 0).$$

Como $r_5 = 0$, o mdc$(88, 25) = r_4 = 1$. □

Abaixo, criamos uma função chamada `mdc_euclides`, que consiste em uma implementação do algoritmo descrito no Teorema 2. Na linha 18 do bloco de códigos, a função é chamada com os argumentos 25 e 88.

Bloco de Códigos 4.

```
1  def mdc_euclides(a,b):
2      if a<1 or b<1:
3          print('a e b precisam ser maiores que zero')
4      else:
5          r0 = max([a,b])             # define r0
6          r1 = min([a,b])             # define r1
7          q=[r0, r0//r1]              # lista para guardar os q's
8          r=[r1, r0%r1]               # lista para guardar os r's
9
10         while r[-1] != 0:           #
11             q.append(r[-2]//r[-1])  # Cálculo de todos os
12             r.append(r[-2]%r[-1])   # valores dos q's e r's
13         q=q[1:]
14         print('%d = %d*%d+%d' %(r0, r1, q[0], r[1]))
15         for i in range(len(q)-1):
16             print('%d = %d*%d+%d' %(r[i], r[i+1], q[i+1], r[i+2]))
17         print('\nmdc(%d,%d)=%d' %(a,b,r[-2]))
18 mdc_euclides(25,88)
```

Saída do Bloco de Códigos 4.

```
88 = 25*3+13
25 = 13*1+12
13 = 12*1+1
12 = 1*12+0

mdc(25,88)=1
```

Uma abordagem didática para o código acima ocorre na criação de problemas relacionados com MDC, nos quais são necessários as exibições dos cálculos intermediários. Tais problemas são comuns em primeiros cursos que trabalhem com Aritmética dos Inteiros ou Teoria dos Números, especialmente nos cursos de Graduação em Matemática. A ferramenta cria inúmeros problemas, com a solução descrita, e em tempo demasiadamente menor do que a criação manual por tentativa e cálculo.

Observação 2. Existe um método para o cálculo do MDC de n inteiros, reduzindo-o ao cálculo do MDC de $n-1$ pares de inteiros. Dados números inteiros $a_1, a_2, \ldots, a_n$, não todos nulos, existe seu MDC e

$$\text{mdc}(a_1, a_2, \ldots, a_n) = \text{mdc}(a_1, a_2, \ldots, \text{mdc}(a_{n-1}, a_n)).$$

O *SageMath* não possui um método que calcule diretamente o MDC para mais de duas entradas, entretanto usando a observação anterior, podemos fazer uma adaptação para que o algoritmo do *software* seja usado recursivamente e assim obtermos o MDC de uma quantidade arbitrária de entradas, como pode ser visto no Bloco de Códigos 5.

A seguir, apresentamos uma função que calcula o MDC para mais de dois números inteiros, utilizando a Observação 2 e o método `gcd` do *SageMath*. Na linha 2, o código verifica se a lista possui menos de duas entradas; caso verdadeiro, uma mensagem de erro será apresentada. Na linha 4, o código verifica se a lista possui exatamente duas entradas; se sim, o algoritmo executará o método `gcd`. Na linha 6, caso a lista tenha mais de dois elementos, o algoritmo passará para a linha 7, onde a `lista2` é criada com a primeira entrada sendo o MDC das duas primeiras entradas da lista original e as outras entradas sendo todas as entradas da lista original, a partir da terceira entrada. Na linha 8, a função chama a si mesma usando a `lista2` como entrada.

Bloco de Códigos 5.

```
1 def mdc_recursivo(lista):
2     if len(lista) < 2:
3         return 'ERRO, a entrada precisa de pelo menos 2 argumentos'
4     elif len(lista) == 2:
5         return gcd(lista)
6     else:
7         lista2  = [gcd(lista[0], lista[1])] + list(lista[2:])
8         return mdc_recursivo(lista2)
9 mdc_recursivo([56, 84, 154, 210])
```

Saída do Bloco de Códigos 5.

```
1 14
```

No Bloco de Códigos a seguir, executamos a função definida no Bloco de Códigos 5 com polinômios nas entradas.

Bloco de Códigos 6.

```
1 n = polygen(ZZ,'n')
2 mdc_recursivo([n^3-n, n^2-n, 3*n-3])
```

Saída do Bloco de Códigos 6.

```
1 n - 1
```

Teorema 3 (Bachet-Bézout)**.** *Se a e b são inteiros, um dos quais não é nulo, então existem inteiros x e y tais que $ax + by = \text{mdc}(a, b)$.*

Demonstração. Veja página 80 de [2] ou página 20 de [3]. □

Exemplo 3. Encontre valores para x e y de modo que

$$88x + 25y = \text{mdc}(88, 25) = 1.$$

Solução. Vamos utilizar os passos da solução do Exemplo 2 na ordem inversa. Assim, para $i = 3$ podemos escrever

$$1 = 13 - 12 \times 1.$$

Substituindo o valor de 12 obtido no passo quando $i = 2$, obtemos:

$$1 = 13 - (25 - 13 \times 1) \times 1.$$

Organizando a expressão anterior ficamos com

$$1 = 13 \times 2 - 25.$$

Substituindo o valor de 13 obtido no passo quando $i = 1$, obtemos:

$$1 = (88 - 25 \times 3) \times 2 - 25.$$

Organizando, ficamos com

$$1 = 88 \times 2 + 25 \times (-7).$$

Assim, $x = 2$ e $y = -7$. □

Os valores do MDC, x e y, tais que $ax + by = \text{mdc}(a, b)$ (conforme o Teorema 3), também podem ser facilmente calculados no *SageMath*, basta usar o comando `xgcd(a,b)`, este retornará a tripla $(\text{mdc}(a, b), x, y)$. Veja o código a seguir:

Bloco de Códigos 7.

```
xgcd(88, 25)
```

Saída do Bloco de Códigos 7.

```
(1, 2, -7)
```

A seguir, apresentamos uma nova função que nomeamos de `mdc_bachet_bezout`. Diferente do método `xgdc` do *SageMath*, a nossa função nos retorna todos os passos intermediários no cálculo de x e y, de forma que satisfaçam a equação $ax + by = \text{mdc}(a, b)$, semelhante ao que foi feito no Exemplo 3. Para simplificar a função, aplicamos o algoritmo derivado do Teorema 2 (uma modificação do código apresentado no Bloco de Códigos 4) para utilizar a lista dos restos obtidos. Com essa lista, utilizamos o método `xgcd` do *SageMath* para exibir cada etapa do processo.

Bloco de Códigos 8.

```
def mdc_bachet_bezout(a,b):
    if 1>a or 1>b:
        print('a e b precisam ser maiores que zero')
    else:
        r0 = max([a,b])             # define r0
        r1 = min([a,b])             # define r1
        q=[r0, r0//r1]              # lista para guardar os q's
        r=[r1, r0%r1]               # lista para guardar os r's

        while r[-1] != 0:           #
```

```
            q.append(r[-2]//r[-1])  # cálculo de todos os
            r.append(r[-2]%r[-1])   #  valores dos q's e r's

        r.reverse()                   # lista dos r's na ordem invertida
        r=r[1:]+[max([a,b])]          # lista dos r's da entrada 1 em diante
                                      #  adicionada do maximo entre a e b

        for i in range(len(r)-2):     # processo de impressão na tela usando
            x=xgcd(r[i+1],r[i+2])     #  o método xgcd
            print('%d = %d*(%d) + %d*(%d)' %(r[0],r[i+1],x[1],r[i+2],x[2]))
mdc_bachet_bezout(25,88)
```

Saída do Bloco de Códigos 8.

```
1 = 12*(-1) + 13*(1)
1 = 13*(2) + 25*(-1)
1 = 25*(-7) + 88*(2)
```

3. Sobre Equações Diofantinas Lineares

Nesta seção formalizaremos os conceitos já citados na introdução sobre as Equações Diofantinas Lineares, com o intuito de apresentar, em seguida, o tema principal deste trabalho: O Número de Frobenius.

Seja $f : \mathbb{Z}^n \to \mathbb{Z}$ uma função polinomial multilinear[2] a coeficientes inteiros, onde $n \geq 2$. Uma Equação Diofantina Linear é uma equação da forma

$$f(x_1, x_2, ..., x_n) = c, \tag{1}$$

para algum c. Uma solução da Equação (1) é um conjunto de n-uplas $(\alpha_1, \alpha_2, ..., \alpha_n) \in \mathbb{Z}^n$ tal que $f(\alpha_1, \alpha_2, ..., \alpha_n) = c$, isto é, para cada $c \in \mathbb{Z}$, o conjunto solução de (1) é dado por $f^{-1}(c)$. Equivalentemente, a Equação (1) pode ser reescrita como uma combinação linear dos números inteiros $x_1, x_2, ..., x_n, c$ de tal forma que

$$a_1x_1 + a_2x_2 + ... + a_nx_n - c = 0, \tag{2}$$

onde a_1, a_2,...,a_n são números inteiros.

Interpretando geometricamente, encontrar uma solução da equação acima é equivalente a encontrar os pontos do hiperplano[3] que contém coordenadas inteiras cuja equação é dada pela Equação (2), para cada c. Em particular, se $n = 2$ e usando a Equação (2), temos a definição abaixo:

Definição 3. Sejam a, b e c números inteiros. As equações do tipo

$$ax + by = c$$

para as quais só se está interessado em soluções inteiras, são chamadas de *Equações Diofantinas Lineares de duas variáveis ou dimensão 2*.

[2] Aquela em que é linear para cada uma das variáveis.
[3] Um plano em dimensão $n > 2$.

O próximo resultado mostra uma condição necessária e suficiente para garantir que uma Equação Diofantina Linear admita solução.

Proposição 1. *A Equação Diofantina Linear dada por* $ax + by = c$ *admite solução em números inteiros se, e somente se,* $\text{mdc}(a, b)|c$.

Demonstração. Veja página 100 de [2]. □

Sabemos como obter uma solução particular da Equação Diofantina $ax + by = c$, basta usar o procedimento exibido nos Exemplos 2 e 3.

A proposição a seguir mostra como obter a solução geral a partir de uma solução particular.

Proposição 2. *Seja* (x_0, y_0) *uma solução particular da Equação Diofantina Linear* $ax + by = c$, *na qual,* $\text{mdc}(a, b) = 1$. *Então, o seu conjunto solução, é composto pelos pares* (x, y) *em* $\mathbb{Z}^2$ *tais que*

$$x = x_0 + tb, \quad y = y_0 - ta; \quad t \in \mathbb{Z}.$$

Demonstração. Veja página 101 de [2]. □

Observação 3. Perceba que a solução está parametrizada em função do parâmetro t. Isolando este parâmetro, descobrimos que x e y satisfazem a seguinte equação

$$\frac{y - y_0}{a} = t = \frac{x - x_0}{b},$$

ou organizando

$$y = \frac{a}{b}x + \frac{by_0 - ax_0}{b}. \tag{3}$$

Isso significa que as soluções pertencem a reta da Equação (3).

No exemplo a seguir, temos uma aplicação da Proposição 2, onde estamos interessados em detectar o conjunto solução que possui coordenadas inteiras e positivas da *Equação Diofantina*.

Exemplo 4. Em uma situação hipotética, as cédulas de dinheiro de um caixa eletrônico disponíveis para saque são de \$3 e de \$8 apenas. Supondo que o caixa eletrônico possua cédulas suficientes:

(a) É possível sacar \$13?

(b) A partir de que valor, sempre é possível sacar uma determinada quantidade de dinheiro nessa máquina?

Solução. **(a)** Vamos usar as Proposições 1 e 2 para mostrar que o item **(a)** admite solução nos inteiros, mas não admite solução nos naturais.

Observe que $\text{mdc}(3, 8) = 1$, pela Proposição 1, como $\text{mdc}(3, 8)|13$, a equação $3x + 8y = 13$ admite solução nos inteiros. Aplicando o algoritmo descrito no Teorema 2, obtemos o valor do $\text{mdc}(3, 8) = 1$:

$$\begin{aligned} 8 &= 3 \cdot 2 + 2 \\ 3 &= 2 \cdot 1 + 1 \\ 2 &= 1 \cdot 2 + 0 \end{aligned}$$

Agora, usando o Teorema 3, na qual a ideia para obter os valores para x e y é apresentada no Exemplo 3, podemos escrever o valor do mdc$(3, 8)$ como

$$\begin{aligned} 1 &= 3-2 \\ 1 &= 3-(8-3\cdot 2) \\ 1 &= 3-8+3\cdot 2 \\ 1 &= 3\cdot 3+8\cdot(-1). \end{aligned}$$

Para obter uma solução para a Equação Diofantina $3x+8y = 13$, basta multiplicar a igualdade $3\cdot3+8\cdot(-1) = 1$, por 13. Assim, obtemos

$$3\times 39+8\times(-13) = 13.$$

Aplicando a Proposição 2, a solução geral é dada por

$$(x, y) = (39+8t, -13-3t), \quad t \in \mathbb{Z}.$$

Usando a solução geral, para que exista solução em $\mathbb{N}$ é necessário que

$$39+8t \geq 0 \quad \text{e} \quad -13-3t \geq 0.$$

Resolvendo as inequações obtemos

$$t \geq -4,875 \quad \text{e} \quad t \leq -4,\overline{3}.$$

Portanto, não existe solução em $\mathbb{N}$.

Para o item **(b)**, note que não é possível obter o valor 13, mas será que é possível obter o valor 14? E os seguintes? De fato:

$$\begin{aligned} 3\times(2+t)+8\times 1 &= 14+3t \\ 3\times(5+t)+8\times 0 &= 15+3t \\ 3\times(0+t)+8\times 2 &= 16+3t \end{aligned}$$

Fazendo $t \in \mathbb{Z}_+$, construímos uma parametrização para obter qualquer valor maior ou igual a 14 a partir da expressão $3x+8y$, com $x, y \geq 0$. □

Note que a solução apresentada para o item **(b)** não envolve um método geral, foi apenas uma solução particular para o problema específico da expressão $3x+8y$. O caso geral será tratado na próxima seção.

O *SageMath* possui o método `solve_diophantine`, específico para resolver equações diofantinas. Abaixo, utilizando este método, calculamos a solução da Equação Diofantina $3x+8y = 13$.

Bloco de Códigos 9.

```
x,y=var('x, y')
solve_diophantine(3*x+8*y==13)
```

Saída do Bloco de Códigos 9.

```
(8*t_0 + 39, -3*t_0 - 13)
```

O método `solve_diophantine` no *SageMath* também funciona para algumas equações diofantinas não lineares, como pode ser visto no Bloco de Códigos a seguir o qual resolve o problema das Ternas Pitagóricas:

Bloco de Códigos 10.

```
x,y,z = var('x,y,z')
solve_diophantine(x^2+y^2==z^2)
```

Saída do Bloco de Códigos 10.

```
(2*p*q, p^2 - q^2, p^2 + q^2)
```

Perceba que a solução coincide com a famosa solução parametrizada das Ternas Pitagóricas.

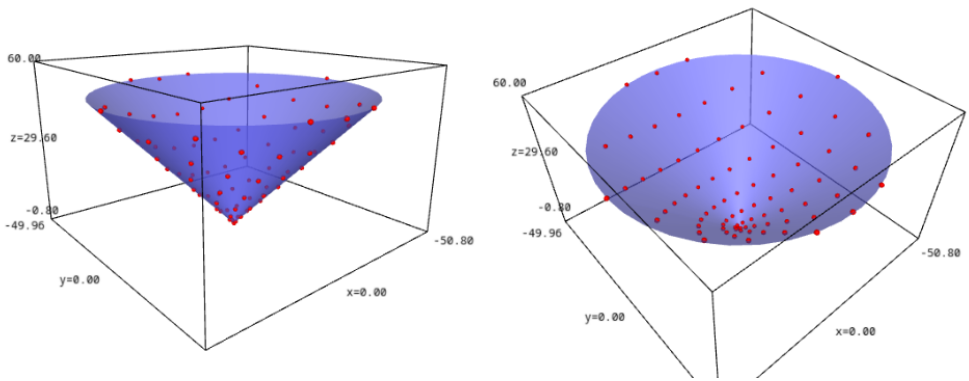


Figura 4: Representação gráfica do conjunto solução de ternas pitagóricas (x, y, z) de inteiros positivos (na cor vermelha) satisfazendo $z = \sqrt{x^2 + y^2}$ (na cor azul), vista de dois ângulos. A parte em azul é o conjunto solução com $x, y, z \in \mathbb{R}$ e os pontos em vermelho são as soluções com $x, y, z \in \mathbb{Z}$.

Abaixo, desenvolvemos uma função para calcular algumas soluções e gerar um gráfico para qualquer Equação Diofantina Linear em dimensão 2.

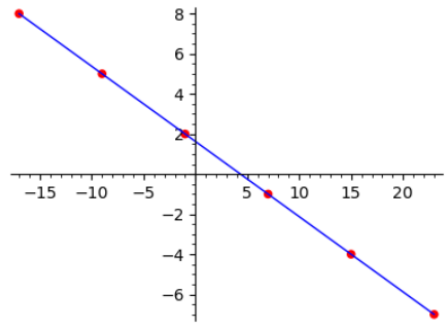
Na linha 25, a função é chamada para resolver e esboçar o gráfico da Equação Diofantina $3x + 8y = 13$.

Bloco de Códigos 11.

```
def graf_diof_lin(a,b,c,n_sols=6):
    x,y,t_0=var('x,y,t_0')
    if c%gcd(a, b)!=0:   #verifica se o mdc(a,b) não divide c
        print('O mcd(%d, %d) não divide %d' %(a, b, c))  #caso divida, mens. de erro
    else:   #caso mdc(a,b)|c, o código continua na linha 6
        l=[]   #cria a lista vazia l, onde guardaremos as soluções inteiras
        sd = solve_diophantine(a*x+b*y==c)  #obtendo a solução geral da eq. diof.
        t0 = solve(sd[0]==0, t_0)  #obtendo o valor de t_0 que a reta cruza o eixo y
        t0 = ceil(t0[0].rhs())     #arredondando t0 para um valor inteiro
        for i in range(t0, t0+ceil(n_sols/2)):
            #guarda soluções inteiras depois de t0
            l.append((sd[0].subs(t_0=(i)), sd[1].subs(t_0=(i))))
```

```
        for i in range(t0-1, t0-ceil(n_sols/2)-1, -1):
            #guarda soluções inteiras antes de t0
            l.append((sd[0].subs(t_0=(i)), sd[1].subs(t_0=(i))))
        l.sort()   #ordena l
        ps = l[0]  #guarda em ps a primeira solução da lista l
        us = l[-1] #guarda em us a última solução da lista l
        p2=line((ps, us))  #gráfico da reta que liga ps até us
        p1=list_plot(l, size=30, color='red') #gráfico com as soluções inteiras
        (p2+p1).show(figsize=4) #exibe os gráficos guardados em p2 e p1 sobrepostos
        print('As soluções destacadas de %dx + %dy = %d são:' %(a, b, c))
        for i in l:
            print(i)  #exibe as soluções inteiras
graf_diof_lin(3, 8, 13)
```

Saída do Bloco de Códigos 11.




```
As soluções destacadas de 3x + 8y = 13 são:
(-17, 8)
(-9, 5)
(-1, 2)
(7, -1)
(15, -4)
(23, -7)
```

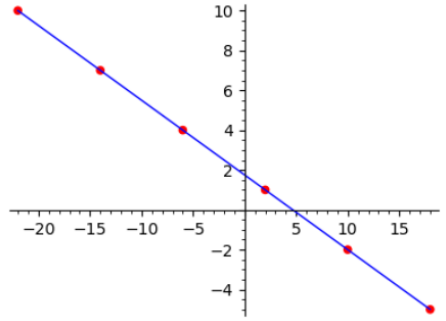
Figura 5: Sem solução com x e y inteiros e maiores ou iguais a zero.

No bloco de códigos a seguir, executamos a função definida no Bloco de Códigos 11, trocando o último argumento da função, de 13 para 14.

Bloco de Códigos 12.

```
graf_diof_lin(3, 8, 14)
```

Saída do Bloco de Códigos 12.




```
As soluções destacadas de 3x + 8y = 14 são:
(-22, 10)
(-14, 7)
(-6, 4)
(2, 1)
(10, -2)
(18, -5)
```

Figura 6: Uma solução com $x = 2$ e $y = 1$.

No gráfico acima, podemos ver explicitamente que o item **(a)** do Exemplo 4 não possui solução, pois as soluções precisam estar presentes no primeiro quadrante do plano cartesiano.

Problemas como o Exemplo 4 são relativamente comuns no estudo das Equações Diofantinas Lineares. Eles fazem parte do famoso Problema das Moedas de Frobenius, cuja temática será desenvolvida na próxima seção.

4. O Número de Frobenius

O problema do troco de Frobenius ou o problema de Frobenius é interpretado algebricamente da seguinte maneira:

Suponha que existem moedas denominadas por $m_1, m_2, \ldots, m_n$. Qual o menor valor que pode ser pago utilizando tais moedas? Traduzindo de forma algébrica, temos:

Sejam $m_1, m_2, \ldots, m_n$ números inteiros positivos. Determinar o menor inteiro positivo $T = T(m_1, m_2, \ldots, m_n)$ tal que todo inteiro positivo $a \geq T$ pode ser rescrito como combinação inteira positiva de $m_1, \ldots, m_n$, isto é, a equação

$$m_1x_1 + m_2x_2 + \ldots + m_nx_n = a$$

possui solução não negativa, isto é, $x_i \geq 0$ para $i = 1, 2, \ldots, n$.

O teorema abaixo que trata do Número de Frobenius, também é conhecido pelo Teorema das Moedas ou do Troco de Frobenius.

Teorema 4 (Número de Frobenius)**.** *Sejam a e b inteiros positivos com* $\text{mdc}(a, b) = 1$. *Considere a expressão $ax + by$, com $x, y \in \mathbb{Z}_+$. O maior inteiro $g = g(a, b)$, tal que $ax + by = g$ não admite solução, mas $ax + by = d$ admite solução sempre que $d \in \mathbb{Z}$ e $d > g$, é dado por*

$$g = ab - a - b.$$

Demonstração. Veja página 341 de [9]. □

A seguir, desenvolvemos uma função que calcula o número de Frobenius (de acordo com o Teorema 4) e exibe uma parametrização das soluções das equações diofantinas em $\mathbb{Z}_+$, com o valor constante maior que o número de Frobenius, da mesma maneira que o item **(b)** do Exemplo 4. Na linha 2 a função verifica se $\text{mdc}(a, b) \neq 1$, caso isto verdadeiro, a função pede para trocar as entradas de forma que o MDC seja igual a 1. No caso em que $\text{mdc}(a, b) = 1$ o número de Frobenius é calculado na linha 5 e guardado na variável `g`. Da linha 6 até a linha 17 o código organiza as informações para imprimir o texto na tela sobre a Equação Diofantina e o número de Frobenius. Da linha 19 até a linha 23 o algoritmo procura pela solução de cada Equação Diofantina $ax + by = g + i$ com i variando de zero até o $\min\{a, b\}$. Na linha 29, a função é chamada com os argumentos 3 e 8.

Bloco de Códigos 13.

```
1  def num_frob(a,b):
2      if gcd(a, b)!=1:
3          print('Insira valores tais que o mdc seja igual a 1.')
4      else:
5          g = a*b-a-b
6          print('g = %d*%d-%d-%d=%d' %(a, b, a, b, g))
7          m = min([a,b])
8          M = max([a,b])
9          l=[]
10         dic={}
11         temp=0
12
13         print('Abaixo, sendo t inteiro não negativo, temos uma')
14         print('parametrização para obter qualquer valor inteiro,')
15         print('maior ou igual a %d que pode ser obtido pela' %(g+1))
16         print('expressão: %dx+%dy.\n' %(m, M))
17
18         for j in range(m):
19             for i in range(m):
20                 temp = solve(m*x + M*j==g+i+1, x)[0].rhs()
21                 if temp in ZZ:
22                     dic.update({(g+i+1):[temp, j]})
23
24         for j in range(g+1, g+m+1):
25             print('%d*(%d+t) + %d*%d = %d + %d*t' %(m,dic[j][0],M,dic[j][1],j,m))
26 num_frob(3,8)
```

Saída do Bloco de Códigos 13.

```
g = 3*8-3-8=13
Abaixo, sendo t inteiro não negativo, temos uma
parametrização para obter qualquer valor inteiro,
maior ou igual a 14 que pode ser obtido pela
expressão: 3x+8y.

3*(2+t) + 8*1 = 14 + 3*t
3*(5+t) + 8*0 = 15 + 3*t
3*(0+t) + 8*2 = 16 + 3*t
```

Abordagens didáticas relacionadas ao Bloco de Código 14 permitem tanto a conferência de soluções de problemas semelhantes, quanto a criação de uma infinidade de novos, uma vez que a saída do código determina tanto o menor valor a qual há solução em inteiros positivos quanto exibe as soluções.

4.1. Uma proposta de extensão do Número de Frobenius para "três moedas"

Como vimos, o número de Frobenius para "duas moedas" é fácil de ser calculado, e possui uma expressão explícita. No entanto, uma fórmula (simples) para o número de Frobenius com "três moedas" é muito mais difícil, e tem sido o assunto de vários trabalhos. Frank Curtis provou que a busca por tal fórmula está, em certo sentido, fadada ao fracasso. Dessa forma, apresentaremos uma proposta de extensão do que fizemos na seção anterior, considerando agora o problema do número de Frobenius com três "moedas".

O que foi provado é que para alguns casos é possível fazer este cálculo de maneira simples, obtendo até uma expressão explícita, como pode ser visto no teorema a seguir:

Teorema 5 (Teorema 2.3.2 de [1])**.** *Sejam* a_1, a_2, a_3 *inteiros com* $\text{mdc}(a_1, a_2, a_3) = 1$. *Se* $\text{mdc}(a_1, a_2) = d$, $a_1 = a_1' d$, $a_2 = a_2' d$ *e existem inteiros* $x_1, x_2 \geq 0$ *com* $a_3 = x_1 a_1' + x_2 a_2'$. *Então,*

$$g(a_1, a_2, a_3) = \frac{a_1 a_2}{d} - a_1 - a_2 + (d-1)a_3.$$

O livro [1] possui diversos algoritmos e teoremas sobre o número de Frobenius para mais de duas "moedas". Considerando o caso para três "moedas", utilizamos o Algoritmo de Rødseth (página 3) e o Teorema 2.3.2 para formular um *widget* que está disponível no site complementar, cujo *link* está disponível na introdução deste artigo. Apresentaremos os códigos envolvidos na criação deste *widget* que é uma proposta nossa, e que inclui os resultados citados. Nosso algoritmo identifica as situações listadas e transita entre os resultados para obter o número de Frobenius para três "moedas". Dada a complexidade deste caso, o algoritmo é mais extenso, e as explicações serão deixadas nas linhas de comando.

Bloco de Códigos 14.

```
def teste_teo_2_3_1(a,b):                    #
    a1 = min([a,b])                          # calcula os valores
    b1 = max([a,b])                          # que não podem ser
    l=[]                                     # obtidos por
    for y in range(1,a1):                    # a1'*x1+a2'*x2 com
        for i in range(1,int(y*b1/a1)+1):    # x, y >= 0
            l.append(b1*y-a1*i)              #
    return l                                 #
def frobenius_3(a1, a2, a3):

    d = gcd(a1, gcd(a2,a3))
    if d>1:
        return 'Escolha 3 números relativamente primos'

    d=gcd(a1, a2)       # caso mdc(a1,a2)>1
    if d>1:             # precisamos aplicar
        a1=a1/d         # Theorem 2.3.1, pag. 36
        a2=a2/d         # da referência

```

```
    lista_n_sol =  teste_teo_2_3_1(a1, a2)
    if a3 not in lista_n_sol:
        g = d*(a1*a2-a1-a2)+(d-1)*a3
        print(f'Pelo Teorema 2.3.2, g = a1*a2/d - a1 - a2 + (d-1)*a3')
        return print(f'g({a1*d},{a2*d},{a3}) = {g}')

    ### Algoritmo de Rodseth ###
    s0 = ZZ(solve_mod([a2*x==a3], a1)[0][0])   #cálculo do s0

    s={0:s0}   # cria o dicionário dos si's
    q={}       # cria o dicionário dos qi's

    q.update({1:((int(a1/s[0]))+1)})       # adiciona q1 no dic q
    s.update({1:(q[1]*s[0]-a1)})           # adiciona s1 no dic q

    i=1                                                 #
    while s[i]!=0:                                      #
        if (int(s[i-1]/s[i]))!= (s[i-1]/s[i]):          # calcula e guarda
            q.update({(i+1):(int((s[i-1]/s[i]))+1)})    # nos respectivos
            s.update({(i+1):(q[i+1]*s[i]-s[i-1])})      # dicionários
            i=i+1                                       # os valores de qi e si
        else:                                           # para i >= 2
            q.update({(i+1):(int(s[i-1]/s[i]))})        #
            s.update({(i+1):(0)})                       #
            i=i+1                                       #

    p = {-1:0, 0:1, 1:(q[1]*1 - 0)}  # cria o dicionário dos pi's com i=-1, 0, 1
    r={0:(s[0]*a2-p[0]*a3)/a1}       # cria o dicionário dos ri's
    if 0>=r[0]:                                         #
        v=0                                             # calcula e guarda
    else:                                               # nos respectivos
        i=1                                             # dicionários
        while r[i-1]>0:                                 # os valores de pi e ri
            p.update({i+1:(q[i+1]*p[i] - p[i-1])})      # i>=2 para pi e
            r.update({i:((s[i]*a2-p[i]*a3)/a1)})        # i>=1 para ri
            i+=1                                        #
        v=i-2                                           #

    g = -a1+a2*(s[v]-1) +a3*(p[v+1]-1) - min([a2*s[v+1], a3*p[v]])  # calcula g
    if d==1:              # caso mdc(a1, a2) = 1 o valor de g já está pronto
        print('Pelo algoritmo de Rodseth')
        return print(f'g({a1},{a2},{a3}) = {g}')
    else:                 # caso mdc(a1, a2) > 1 aplicamos Theorem 2.3.1, pag. 36
        print('Pelo algoritmo de Rodseth')
        g = d*g +(d-1)*a3
        return print(f'g({a1*d},{a2*d},{a3}) = {g}')
frobenius_3(45,96,74)
```

Saída do Bloco de Códigos 14.

```
Pelo algoritmo de Rodseth
g(45,96,74) = 901
```

5. Conclusão

Neste artigo, fundamentados na BNCC, investigamos alguns dos temas da Teoria dos Números, na perspectiva de criar algumas tecnologias sob a forma de ferramentas computacionais que permitissem a automatização de processos. As tecnologias permeiam os conteúdos de Divisão Euclidiana, Máximo Divisor Comum, Equações Diofantinas Lineares e o Número de Frobenius. Através da análise desses conceitos, foi possível aprofundar a compreensão dos temas e explorar algumas abordagens didáticas em contextos educacionais. Os códigos desenvolvidos ao longo deste trabalho demonstraram ser ferramentas valiosas tanto para a validação de soluções quanto para a criação de novos desafios matemáticos, por exemplo, o Bloco de Código 3 nos permite a criação de infinitos problemas semelhantes ao do Exemplo 1, extraído da Olimpíada Internacional de Matemática (*IMO*). A automatização desses processos não apenas agiliza a verificação de resultados, mas também permite a geração de problemas com diferentes níveis de dificuldade de maneira eficiente e sistemática. Assim, este trabalho contribui tanto para o entendimento global nos conteúdos estabelecidos quanto no desenvolvimento de tecnologias computacionais para a criação e verificação da solução de problemas da Teoria dos Números. Além disso, por meio das abordagens didáticas estabelecidas, acreditamos que o trabalho permite um relacionamento direto entre o Ensino e Aprendizagem no tema proposto. Por fim, disponibilizamos também um material *online* que contém todas as tecnologias desenvolvidas por nós nessa pesquisa, e que já está pronto para ser utilizado, não necessitando de uma compreensão grande de programação para a sua utilização. Esperamos que o trabalho inspire novos desenvolvimentos análogos entre matemática e tecnologia, promovendo um aprendizado mais dinâmico de modo que a utilização das ferramentas computacionais auxiliem como recursos incentivadores do aprendizado, do ensino, da resolução e desenvolvimento de problemas matemáticos.

Referências

[1] ALFONSÍN, JORGE L. RAMÍREZ. **The diophantine Frobenius problem.** OUP Oxford, 2005.

[2] HEFEZ, A. **Aritmética**. Coleção PROFMAT, SBM, 2ed., 2016.

[3] LEMOS, M. **Criptografia, Números Primos e Algoritmos**. IMPA, 4ed., 2010.

[4] SILVA, L. D.; SANTOS, M. P.; MACHADO J. R. N., **Elementos de Computação Matemática com *SageMath***, SBM, 2019.

[5] https://wiki.sagemath.org/interact/. Acessado em 11/04/2024.

[6] https://doc.sagemath.org/html/pt/tutorial/introduction.html#objetivos-do-sage-a-longo-prazo. Acessado em 11/04/2024.

[7] https://doc.sagemath.org/html/pt/tutorial/index.html. Acessado em 11/04/2024.

[8] BRASIL. Ministério da Educação. Base Nacional Comum Curricular. Brasília, 2018. Disponível em: http://basenacionalcomum.mec.gov.br/. Acesso em: 25 de abril de 2024.

[9] NASCIMENTO, E. C. A.; TANAKA, T. Y.; SILVA, B. C., **Equações diofantinas lineares e não lineares: uma abordagem por meio de questões de Olimpíadas de Matemática**. PMO v. 10, n.3, 2022.

[10] PAULO, G. S.; VIEZEL, C.; CORRÊA, I. L., **Python™ com o Google Colab para entender conceitos básicos de Cálculo Diferencial e Integral**. PMO v. 11, n.4, 2023.

[11] SANTOS, P. R.; OLIVEIRA, I. R. C.; FREIRE, E. R. C. G., **O pacote Shiny na criação de applets para o ensino de Funções Matemáticas e Estatıística Descritiva na Educação Básica**. PMO v. 10, n.1, 2022.

[12] SILVA, A. A. R.; DIDIER, M. A. C.; BONFIM, S. C., **Construção de Modelos Matemáticos com Python: Uma Motivação para o Estudo de Cálculo Diferencial e Lógica Matemática**. ReviSem n.1, 2024.

Ricardo N. Machado Junior
Universidade Federal Rural de Pernambuco
<ricardo.machadojunior@ufrpe.br>

Thiago Yukio Tanaka
Universidade Federal Rural de Pernambuco
<thiago.tanaka@ufrpe.br>

Jogli Gidel da Silva Araújo
Universidade Federal Rural de Pernambuco
<jogli.silva@ufrpe.br>

Recebido: 22/05/2024
Publicado: 03/12/2024