UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

THIAGO FELIPE CASTRO CARRENHO

Novas aplicações da representação matricial de partições

Campinas
11/01/2021

THIAGO FELIPE CASTRO CARRENHO

Novas aplicações da representação matricial de partições*

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado, sob a orientação do(a) Prof. José Plinio de Oliveira Santos.

*Este trabalho foi financiado pelo CNPq, projeto 135119/2020.

Resumo

Este projeto tem por finalidade apresentar a implementação de um algoritmo que calcule o número de partições irrestritas de $n \in \mathbb{N}$ a partir de uma bijeção entre o conjunto de partições irrestritas de n e um conjunto de matrizes de duas linhas de inteiros não negativos.

Palavras-Chave: Partições irrestritas. Matrizes de duas linhas.

Abstract

This project aims to present an implementation of an algorithm that computes the number of irrestrict partitions of $n \in \mathbb{N}$ from a bijection between the set of irrestrict partitions of n and a set of two-line matrices of non-negative integers.

Keywords: Irrestrict partitions. Two-line matrices

Conteúdo

1 Introdução **6**

2 Desenvolvimento **8**

2.1 Cálculo de $T_j^{(\ell)}$ 9

2.2 Cálculo de $\mathbb{D}_\ell$ 10

2.3 Algoritmo final 12

3 Resultados **12**

4 Conclusão **15**

5 Anexos **16**

5.1 Implementação do cálculo de $T_j^{(\ell)}$ 16

5.2 Implementação da função soma 16

5.3 Implementação do cálculo de $\mathbb{D}_\ell$ 17

5.4 Implementação da função principal: cálculo de $p(n)$ 17

Bibliografia **19**

1 Introdução

Partindo de algumas definições, vamos encontrar a bijeção entre o conjunto de partições irrestritas de n e um conjunto de matrizes de duas linhas de inteiros não negativos.

Definição 1.1 (Partição)**.** *Uma partição de $n \in \mathbb{N}$ é uma coleção de inteiros positivos cuja soma é n, isto é, $\Omega = \{c_1, c_2, \cdots, c_s\}$ é uma partição de n se $c_1 + c_2 + \cdots + c_s = n$. A ordem das partes $c_1, c_2, \cdots, c_s$ é irrelevante, por isso podemos pedir uma ordenação específica. Por questão de facilidade, vamos pedir que $c_1 \geq c_2 \geq \cdots \geq c_s \geq 1$.*

Definição 1.2 (Conjunto de partições)**.** *$\mathbb{P}(n)$ é o conjunto de partições de n se*

$$\mathbb{P}(n) = \{\Omega = \{c_1, c_2, \cdots, c_s\} | c_1 + c_2 + \cdots + c_s = n\}$$

E o número de partições de n é $p(n) = \#\mathbb{P}(n)$.

Definição 1.3. *$\mathbb{M}(n)$ é o conjunto de todas matrizes de duas linhas do tipo*

$$M = \begin{pmatrix} a_1 & a_2 & \cdots & a_s \\ b_1 & b_2 & \cdots & b_s \end{pmatrix}$$

onde $a_j, b_j \in \mathbb{N} \cup \{0\}$ e

$$a_s = 1,\ a_j = a_{j+1} + b_{j+1} \text{ e } \sum_{i=1}^{s}(a_i + b_i) = n$$

Esta definição é um caso particular de uma definição mais geral $\mathbb{M}(n, c, \lambda)$, apresentada em Godinho and Santos [2020] o caso em que $c = 1$ e $\lambda = 0$, que é o caso específico para partições irrestritas.

Tendo uma matriz de duas linhas desta forma, tomemos $c_j = a_j + b_j$, como definido $a_j, b_j \in \mathbb{N} \cup \{0\}$ e $a_j = a_{j+1} + b_{j+1}\ \forall j = 1, \cdots, s$, temos que $c_j \geq c_{j+1}$, isto é, $\{a_1 + b_1, a_2 + b_2, \cdots, a_s + b_s\}$ é uma partição de n. Dessa forma, vemos que cada matriz se relaciona a uma única partição.

Agora, tendo uma partição $\{c_1, c_2, \cdots, c_s\}$, tomemos $a_s = 1$ e $b_s = c_s - 1$ para a última coluna, e as outras colunas com $a_j = a_{j+1} + b_{j+1}$ e $b_j = c_j - a_j$, para $j = 1, \cdots, s-1$, obtendo uma matriz pertencente a $\mathbb{M}(n)$.

Além disso, é fácil perceber que, por estes dois processos, uma matriz M sempre será ligada a uma partição Ω, e esta partição sempre será ligada à mesma matriz M. Assim, temos uma bijeção entre o conjunto de partições $\mathbb{P}(n)$ e o conjunto $\mathbb{M}(n)$.

Exemplo 1.1. *Tomando $n = 5$, listamos todas as partições e as respectivas matrizes abaixo*

$c_1 + c_2 + \cdots + c_s$	$\{c_1, c_2, \cdots, c_s\}$	$\begin{pmatrix} a_1 & a_2 & \cdots & a_s \\ b_1 & b_2 & \cdots & b_s \end{pmatrix}$
5	$\{5\}$	$\begin{pmatrix} 1 \\ 4 \end{pmatrix}$
$4+1$	$\{4,1\}$	$\begin{pmatrix} 1 & 1 \\ 3 & 0 \end{pmatrix}$
$3+2$	$\{3,2\}$	$\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$
$3+1+1$	$\{3,1,1\}$	$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 0 & 0 \end{pmatrix}$
$2+2+1$	$\{2,2,1\}$	$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$
$2+1+1+1$	$\{2,1,1,1\}$	$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$
$1+1+1+1+1$	$\{1,1,1,1,1\}$	$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$

Assim, podemos ver também que $p(5) = 7$.

Uma explicação mais detalhada sobre partições pode ser encontrada em Santos et al. [2007], para saber mais sobre a representação de partições em matrizes de duas linhas, veja Santos et al. [2011], e uma visão de um procedimento por caminhos, e uma aproximação obtida a partir deste procedimento são abordados em Santos and Matte [2018].

Obtendo a bijeção, temos que a cardinalidade dos dois conjuntos é a mesma, e

este algoritmo calcula quantas matrizes existem no conjunto $\mathbb{M}(n)$, que, por haver bijeção com $\mathbb{P}(n)$, é o mesmo que o número de partições irrestritas de n.

Partindo disso, a dedução do algoritmo depende da análise feita no artigo Godinho and Santos [2020], que possui um caso especial para partições irrestritas, uma fórmula fechada para este caso, e é este algoritmo que buscamos implementar.

2 Desenvolvimento

Dado $n \in \mathbb{N}$, queremos que o algoritmo nos devolva $\#\mathbb{M}(n)$, isto é, $p(n)$. Para tal, Godinho and Santos [2020] define blocos de matrizes e descendentes de cada um destes blocos, o artigo é baseado no caso com algumas restrições ($\mathbb{M}(n, c, \lambda)$), mas no caso irrestrito (lembrando que $\mathbb{M}(n) = \mathbb{M}(n, 1, 0)$), sabemos quantos e quais são os blocos:

$$B(0) = \begin{pmatrix} 1 \\ n-1 \end{pmatrix}; B(1) = \begin{pmatrix} 1 & 1 \\ n-2 & 0 \end{pmatrix}; \cdots; B(n-1) = \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \end{pmatrix}}_{n \text{ colunas}}$$

Perceba que $B(j)$ tem $j+1$ colunas, isto é, temos n blocos.

Cada um desses blocos tem uma família de descendentes criando uma árvore de matrizes. A maneira de calcular é por geração, isto é, na visão de árvore, contar por linha, o número de matrizes numa mesma linha, este número denotaremos por $\mathbb{D}_\ell$.

Para definir estes valores, precisamos do seguinte polinômio:

$$\mathcal{P}_0(x) = n - 2(x+1)$$

Cuja raiz é $x_0 = \dfrac{n}{2} - 1$, e com esta obtemos $\ell_0 = \left\lfloor \dfrac{n-2}{2} \right\rfloor + 1 = \left\lfloor \dfrac{n}{2} \right\rfloor$, que é o número de gerações da família de matrizes.

Com ℓ_0 podemos definir os seguintes polinômios:

$$\mathcal{P}_1(x) = n - 1 - x \text{ e } \mathcal{P}_j(x) = n - 2j - x \text{ para } j = 2, \cdots, \ell_0,$$

Cujas raízes são:

$$r_1 = n - 1 \text{ e } r_j = n - 2j \text{ para } j = 2, \cdots, \ell_0.$$

Definamos então $T_j^{(\ell)}$, para $j = 2, \cdots, \ell - 1$, que é um valor necessário para o cálculo de $\mathbb{D}_\ell$, da seguinte forma:

$$T_j^{(\ell)} = T_j^{(\ell)}(t_1, t_2, \cdots, t_{j-1}) = \left\lfloor \frac{\mathcal{P}_\ell(t_1) - \sum_{i=\ell-j+3}^{\ell} i \cdot t_{\ell-i+2}}{\ell - (j-2)} \right\rfloor. \tag{1}$$

Agora podemos definir $\mathbb{D}_1 = r_1 + 1 = n$ (número de blocos, isto é, matrizes da primeira geração), e, usando $T_j^{(\ell)}$ calculamos $\mathbb{D}_\ell$, para $\ell = 2, \cdots, \ell_0$, definido como:

$$\mathbb{D}_\ell = \sum_{t_1=0}^{r_\ell} \sum_{t_2=0}^{T_2^{(\ell)}} \cdots \sum_{t_{\ell-1}=0}^{T_{\ell-1}^{(\ell)}} \left(\left\lfloor \frac{\mathcal{P}_\ell(t_1) - \sum_{i=3}^{\ell} i \cdot t_{\ell-i+2}}{2} \right\rfloor + 1 \right) \tag{2}$$

Por fim, obtido todo $\mathbb{D}_\ell$ para $\ell = 2, \cdots, \ell_0$, temos o número de matrizes em cada geração, logo, o total de matrizes na família é, simplesmente, a soma desses valores:

$$p(n) = \sum_{\ell=1}^{\ell_0} \mathbb{D}_\ell = (r_1 + 1) + \sum_{\ell=2}^{\ell_0} \mathbb{D}_\ell$$

A dedução destas fórmulas está em Godinho and Santos [2020]. Tomando-as definidas como acima, passamos agora a implementar esses cálculos, e o faremos em forma de funções aninhadas.

2.1 Cálculo de $T_j^{(\ell)}$

Partindo de (1), queremos implementar uma função que receba, de entrada, valores de j, ℓ e um vetor $\vec{t} = (t_1, \cdots, t_{j-1})$ e devolva $T_j^{(\ell)}(t_1, \cdots, t_{j-1})$, que deve ser natural.

Para o caso irrestrito, já sabemos a forma explícita de $\mathcal{P}_\ell(x)$, em especial neste

caso podemos substituir na fórmula, o que nos deixa com

$$T_j^{(\ell)} = T_j^{(\ell)}(t_1, t_2, \cdots, t_{j-1}) = \left\lfloor \frac{n - 2\ell - t_1 - \sum_{i=\ell-j+3}^{\ell} i \cdot t_{\ell-i+2}}{\ell - (j-2)} \right\rfloor.$$

Para a implementação da função, podemos perceber que t_i é equivalente a $\vec{t}(i)$, para $i = 1, \cdots, j-1$.

Algoritmo 1: Função $T_j^{(\ell)}(t_1, \cdots, t_{j-1})$

Entrada: $\ell \in \mathbb{N}$, $\ell \geq 2$, $j \in \{2, 3, \cdots, \ell - 1\}$ e $\vec{t} = (t_1, \cdots, t_{j-1})$, vetor de tamanho $j-1$, n

1 $s = 0$;

2 **Para** $i = \ell - j + 3$ *até* ℓ **faça:**

3 $\quad s = s + i \cdot t(\ell - i + 2)$;

4 $T = \dfrac{n - 2\ell - t(1) - s}{\ell - (j-2)}$;

5 $T = floor(T)$ (toma o maior inteiro menor ou igual a T);

Saída: $T_j^{(\ell)}(t_1, \cdots, t_{j-1}) = T$

2.2 Cálculo de $\mathbb{D}_\ell$

Partindo de (2), queremos implementar uma função que receba, de entrada, valores de ℓ e r_ℓ (raiz de $\mathcal{P}_\ell(x)$), e nos devolva o valor de $\mathbb{D}_\ell$.

Para o caso irrestrito, já sabemos a forma explícita de $\mathcal{P}_\ell(x)$, em especial neste caso podemos substituir na fórmula, o que nos deixa com

$$\mathbb{D}_\ell = \sum_{t_1=0}^{r_\ell} \sum_{t_2=0}^{T_2^{(\ell)}} \cdots \sum_{t_{\ell-1}=0}^{T_{\ell-1}^{(\ell)}} \left(\left\lfloor \frac{n - 2\ell - t_1 - \sum_{i=3}^{\ell} i \cdot t_{\ell-i+2}}{2} \right\rfloor + 1 \right).$$

Façamos o somatório por recursão, pois o número de somatórios é variável, para isso, vamos dar como entrada o vetor, que carrega os valores atuais de $t_1, t_2, \cdots, t_\ell$.

Então vamos criar uma função 'soma', que recebe ...

Algoritmo 2: Função soma

Entrada: $\ell \in \mathbb{N}$, $\ell \geq 2$, r_ℓ, $\vec{t} = (t_1, \cdots, t_{\ell-1})$, j, $somatorio$, n

Se $j = 1$ **então**
    $T = r_\ell$;
Senão
    $T = T_j^{(\ell)}(\vec{t})$;
Para $t(j) = 0$ *a* T **faça:**
    Se $j = \ell - 1$ **então**
        $somainterna = 0$;
        Para $i = 3$ *até* ℓ **faça:**
            $somainterna = somainterna + i \cdot t(\ell - i + 2)$;
        $a = \dfrac{n - 2 \cdot \ell - t(1) - somainterna}{2}$;
        $somatorio = somatorio + floor(a) + 1$;
    Senão
        $somatorio = soma(\vec{t}, \ell, r_\ell, somatorio, j + 1, n)$;

Saída: $somatorio$, valor a ser utilizado na função $\mathbb{D}_\ell$

E então a função que calcula $\mathbb{D}_\ell$ fica:

Algoritmo 3: Função $\mathbb{D}_\ell$

Entrada: $\ell \in \mathbb{N}$, $\ell \geq 2$, n, r_ℓ

$j = 1$;
$somatorio = 0$;
$t = (0, 0, \cdots, 0)$ de $\ell - 1$ dimensões;
$somatorio = \text{soma}(\vec{t}, \ell, r_\ell, somatorio, j)$;

Saída: $somatorio = \mathbb{D}_\ell$

2.3 Algoritmo final

Com a função que calcula $\mathbb{D}_\ell$, basta somarmos os valores de $\mathbb{D}_\ell$ para $\ell = 2, \cdots, \ell_0$.

Algoritmo 4: Cálculo de $p(n)$

Entrada: n

$\ell_0 = \left\lfloor \frac{n}{2} \right\rfloor$;

Defina r vetor de ℓ_0 zeros;

$r(1) = n - 1$

$p = r(1) + 1$

Para $\ell = 2$ *até* ℓ_0 **faça:**

    $r(\ell) = n - 2 \cdot \ell$;

    $d = \mathbb{D}_\ell(n, r_\ell)$ usando a função;

    $p = p + d$;

Saída: $p = p(n)$

3 Resultados

O principal resultado é o próprio algoritmo mostrado acima, implementado em funções em MATLAB, as implementações estão no anexo, para não poluir o texto. Fazendo um laço em n dessa função, conseguimos analisar melhor o comportamento do algoritmo.

Além do valor de $p(n)$, coletamos o tempo de processamento do algoritmo para os valores calculados, que foram de $n = 1$ até $n = 167$.

Primeiro de tudo, o gráfico de $n \times p(n)$ comprova que a função $p(n)$ realmente explode para n crescente:

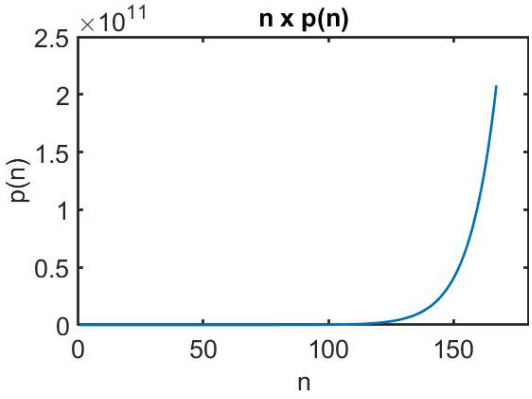
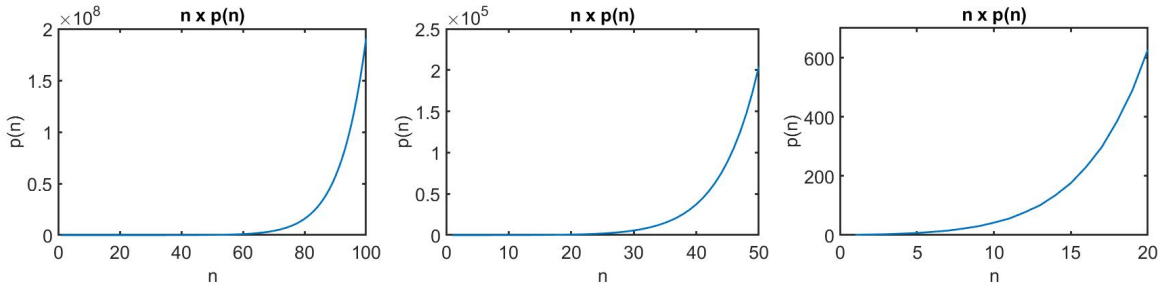


Figura 1: $n \times p(n),\ n = 1 : 167$

Porém, é necessário dividir o gráfico em três menores, pois no Gráfico 1, $p(100)$ aparenta ser um número próximo a 0, quando, na verdade, é da ordem de 10^8. Tomamos, então, os sub-intervalos $1 : 100$, $1 : 50$ e $1 : 20$ e obtivemos os seguintes gráficos:



(a) $n \times p(n),\ n = 1 : 100$ (b) $n \times p(n),\ n = 1 : 50$ (c) $n \times p(n),\ n = 1 : 20$

Figura 2: $n \times p(n)$ para sub-intervalos

Tomando o gráfico log-log com estes mesmos valores, temos o gráfico 3 abaixo, que mostra que o comportamento da função é, não só extremamente crescente, mas mais crescente que exponencial.

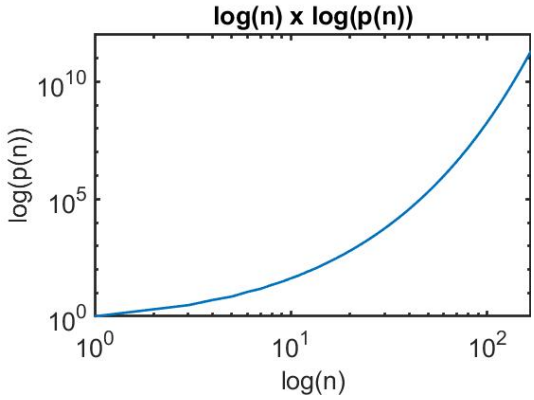


Figura 3: $\log n \times \log p(n),\ n = 1 : 167$

Analisando agora, a velocidade do algoritmo, temos que o tempo gasto para achar cada resultado é aproximadamente proporcional ao próprio resultado $p(n)$, como o gráfico abaixo mostra (aproximadamente uma reta).

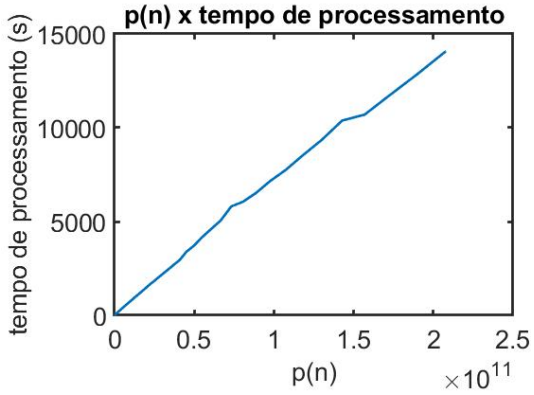


Figura 4: $p(n)\times$tempo de processamento, $n = 1 : 167$

Isto significa que, assim como a função $p(n)$, o tempo de processamento também cresce muito rapidamente, o gráfico abaixo, de n em função do tempo, nos dá uma boa noção deste crescimento.

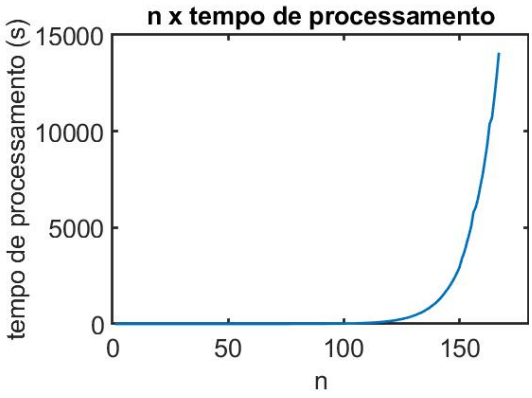


Figura 5: $n\times$tempo de processamento, $n = 1 : 167$

Para simplificar a compreensão, a tabela abaixo mostra intervalos de tempo, e os valores de n cujo processamento se encaixa no intervalo.

Tempo de Processamento	Valores de n
Até 1 segundo	$n = 1 : 76$
De 1 segundo a 1 minuto	$n = 77 : 111$
De 1 minuto até 1 hora	$n = 112 : 151$
Mais de 1 hora	$n = 152 : 167$

Tabela 1: Intervalos de Tempo de Processamento

Com ela, percebe-se que os cálculos se tornam cada vez menos praticáveis para um computador simples.

4 Conclusão

Conclui-se que o algoritmo acima calcula o número de partições irrestritas de n, como era seu objetivo. Conclui-se também que a função $p(n)$ tem um comportamento explosivo, cresce muito rapidamente.

Além disso, é possível concluir que o algoritmo é extremamente eficiente e viável, pois calcula valores até $p(111)$ em menos de um minuto. Entretanto, o algoritmo perde valor para grandes valores de n, já que o tempo se torna impraticável, isto é, o algoritmo tem um 'limite', que depende de quanto tempo o computador pode estar alocado apenas para cálculo desta função.

5 Anexos

5.1 Implementação do cálculo de $T_j^{(\ell)}$

Implementação em MATLAB referente ao algoritmo 1.

Listing 1: Cálculo de $T_j^{(\ell)}$

```
function T = Tj(j,l,t,n)
    s = 0;
    for i = (l-j+3):l
        s = s+i*t(l-i+2);
    end
    T = (n-2*l-t(1)-s)/(l-j+2);
    T = floor(T);
end
```

5.2 Implementação da função soma

Implementação em MATLAB referente ao algoritmo 2.

Listing 2: Função Soma

```
function somatorio = soma(t,l,rl,somatorio,j,n)
    if j == 1
        T = rl;
    else
        T = Tj(j,l,t,n);
    end
    for tj = 0:T
        t(j) = tj;
        if j == l-1
            somainterna = 0;
            for i = 3:l
                somainterna = somainterna + i*t(l-i+2);
            end
```

```
14              a = (n-2*l-t(1)-somainterna)/2;
15              somatorio = somatorio + floor(a) + 1;
16          else
17              somatorio = soma(t,l,rl,somatorio,j+1,n);
18          end
19      end
20  end
```

5.3 Implementação do cálculo de $\mathbb{D}_\ell$

Implementação em MATLAB referente ao algoritmo 2.

Listing 3: Cálculo de $\mathbb{D}_\ell$

```
1  function somatorio = D(l,rl,n)
2      j = 1;
3      somatorio = 0;
4      t = zeros(l-1,1);
5      somatorio = soma(t,l,rl,somatorio,j,n);
6  end
```

5.4 Implementação da função principal: cálculo de $p(n)$

Implementação em MATLAB referente ao algoritmo 4.

Listing 4: Cálculo de $p(n)$

```
1  function p = particao(n)
2      %Valores iniciais
3      lzero = floor(n/2);
4      r = zeros(lzero,1);
5      r(1) = n-1;
6      p = r(1)+1;
7
8      %Laco para somar D2, D3, ..., Dlzero
9      for l=2:lzero
```

```
10          r(l) = n-2*l;
11          d=D(l,r(l),n);
12          p = p+d;
13      end
14  end
```

Bibliografia

Hemar Godinho and José Plínio O. Santos. A family of partitions equinumerous with the set of nodes of a family of trees. *INTEGERS*, 20, 2020. URL http://math.colgate.edu/~integers/u101/u101.pdf.

José Plínio O. Santos and Marilia L. Matte. A new approach to integer partition. *Bulletin of the Brazilian Mathematical Society*, 2018. doi: https://doi.org/10.1007/s00574-018-0082-z.

José Plínio O. Santos, Margarida P. Mello, and Idani T.C. Murari. *Introdução à Análise Combinatória*. Editora Ciência Moderna Ltda., Rio de Janeiro, 2007.

José Plínio O. Santos, Paulo Mondek, and Andréia C. Ribeiro. New two-line arrays representing partitions. *Annals of Combinatorics*, 15(341), 2011. doi: https://doi.org/10.1007/s00026-011-0099-0.